A Layered View Model for XML with Conceptual and Logical Extensions, and its Applications

Submitted by

Rajugan Rajagopalapillai

A thesis submitted in total fulfillment

of the requirements for the degree of

Doctor of Philosophy

Faculty of Information Technology

University of Technology, Sydney (UTS)
Sydney, NSW 2007
Australia

February 2006

CERTIFICATE OF AUTHORSHIP/ORIGINALITY

I certify that the work in this thesis has not previously been submitted for a degree nor has it been submitted as part of requirements for a degree except as fully acknowledged within the text.

I also certify that the thesis has been written by me. Any help that I have received in my research work and the preparation of the thesis itself has been acknowledged. In addition, I certify that all information sources and literature used are indicated in the thesis.

Production Note:
Signature removed prior to publication.

Signature of Author

Date: 26/02/2006

Acknowledgements

I am greatly indebted to my supervisors Professor Tharam Dillon, Professor Elizabeth Chang and Associate Professor Ling Feng for their encouragement and advice, and for providing constant direction and focus to my research. I owe my research achievements to their experienced supervision and unconstraint support.

I thank Professor Tharam S. Dillon for his step-by-step guidance, many hours of research discussions, suggestions, comments and kindhearted support during difficult times. I owe a big thank you to Professor Elizabeth Chang for her many years of unconstraint encouragement and support, patient discussions, motivating comments and advice. I thank Associated Professor Ling Feng for her invaluable suggestions, research discussions and comments on my research ideas and concepts. I am a much better researcher because of my supervisors' excellent guidance.

My thanks also go out to Associate Professor Wenny Rahayu for her help during my research years. I owe special round of thanks to my friends and colleagues at the Faculty of IT, UTS and at the Department of CS-CE, Latrobe University. I also thank Bruna Pomella for commenting ambiguous sentences and correcting grammatical mistakes.

Finally, I owe my success in life to the hard work and sacrifices of the two most important people in my life; Amma and Appa, my parents. I thank them for their unconstraint support and giving me the strength for letting my dream to come true.

Table of Contents

List of Figures

List of Tables

Abstract

EXtensible Markup Language (XML) is becoming the dominant standard for storing, describing and interchanging data among various Enterprises Information Systems (EIS), web repositories and databases. With this increasing reliance on such self-describing, schema-based, semi-structured data language XML, there exists a need to model, design, and manipulate XML and associated semantics at a higher level of abstraction than at the instance level. However, existing OO conceptual modelling languages provide insufficient modelling constructs for utilizing XML structures, descriptions and constraints, and XML and associated schema languages lack the ability to provide higher levels of abstraction, such as conceptual models that are easily understood by humans. To this end, it is interesting to investigate conceptual and schema formalisms as a means of providing higher level semantics in the context of XML-related data modelling. In particular we note that there is a strong need to model views of XML repositories at the conceptual level. This is in contrast to the situation for views for the relational model which are generally defined at the implementation level.

In this research, we use XML view and introduce the Layered View Model (LVM, for short), a declarative conceptual framework for specifying and defining views at a higher level of abstraction. The views in the LVM are specified using explicit conceptual, logical and instance level semantics and provide declarative transformation between these levels of abstraction. For such a task, an elaborated and enhanced OO based modelling and transformation methodology is employed. The LVM framework leads to a number of interesting problems that are studied in this research. First we address the issue of conceptualizing the notion of views: the clear separation of conceptual concerns from the implementation and data language concerns. Here, the LVM views are considered as *first-class citize*ns of the *conceptual model.* Second we provide formal semantics and definitions to enforce representation, specification and definition of such views at the highest level of abstraction, the conceptual level. Third we address the issue of modelling and transformation of LVM views to the required level of abstraction, namely to the schema and instance levels. Finally, we apply LVM to real-world data modelling scenarios to develop other architectural frameworks in the domains such as dimensional XML data modelling, ontology views in the Semantic Web paradigm and modelling user-centred websites and web portals.

List of Publications

During this research, several sections of this research and collaborative works have been published. This includes refereed journal articles, book chapters and refereed international conference papers. Some of these selected published articles (marked with two **) are given in Appendix A.

Journal Articles

1. Rajugan, R., Chang, E, Dillon, TS & Feng, L 2006, 'Modeling Views in the Layered View Model for XML Using UML', *International Journal of Web Information Systems (IJWIS)*, 2006 - vol. 2(2), pp. 95-117.
2. Rajugan, R., Gardner, W, Chang, E & Dillon, TS 2005, 'Designing Websites with EXtensible Web (xWeb) Methodology', *International Journal of Web Information Systems (IJWIS)*, vol. 1(3), pp. 179-91, September.
3. **Nassis, V, Rajugan, R., Dillon, TS & Rahayu, W 2005, 'Conceptual and Systematic Design Approach for XML Document Warehouses', *International Journal of Data Warehousing and Mining*, vol. 1(3), pp. 63-87

Book Chapters

4. Wouters, C, Rajugan, R., Dillon, TS & Rahayu, JW 2006, 'Ontology Extraction Using Views for Semantic Web', in D Taniar & W Rahayu (eds), *Web Semantics and Ontology*, Idea Group Publishing, USA.
5. Nassis, V, Dillon, TS, Rahayu, W & Rajugan, R. 2006, 'Goal-Oriented Requirement Engineering for XML Document Warehouses', in J Darmont & O Boussaid (eds), *Processing and Managing Complex Data for Decision Support*, Idea Group Publishing.

Refereed Conference Papers

6. **Rajugan, R., Chang, E, Feng, L & Dillon, TS 2006, 'Modeling Dynamic Properties in the Layered View Model for XML Using XSemantic Nets', *International Workshop on XML Research and Applications (XRA '06) to be held in conjunction with APWeb '06*, Springer-Verlag, Harbin, China, pp. 142-7.

7. Nassis, V, Dillon, TS, Rajugan, R. & Rahayu, W 2006, 'An XML Document Warehouse Model', *The 11th International Conference on Database Systems for Advanced Applications (DASFAA '06)*, Springer, Singapore - *to appear*.

8. **Rajugan, R., Chang, E, Dillon, TS & Feng, L 2005, 'A Three-Layered XML View Model: A Practical Approach', *24th International Conference on Conceptual Modeling (ER '05)*, Springer-Verlag, Klagenfurt, Austria, pp. 79-95.

9. **Rajugan, R., Chang, E, Dillon, TS & Feng, L 2005, 'Alternate Representations for Visual Constraint Specification in the Layered View Model', *The Seventh International Conference on Information Integration and Web Based Applications & Services (iiWAS '05)*, Austrian Computer Society (OCG), Kuala Lumpur, Malaysia, pp. 571-82.

10. **Rajugan, R., Chang, E, Dillon, TS & Feng, L 2005, 'A Layered View Model for XML Repositories & XML Data Warehouses', *The 5th International Conference on Computer and Information Technology (CIT '05)*, IEEE CS Press, Shanghai, China, pp. 206-15.

11. Rajugan, R., Chang, E, Dillon, TS & Feng, L 2005, 'Engineering XML Solutions Using Views', *The 5th International Conference on Computer and Information Technology (CIT '05)*, IEEE CS Press, Shanghai, China, pp. 116-23.

12. Rajugan, R., Chang, E, Dillon, TS, Feng, L & Wouters, C 2005, 'Modeling Ontology Views: An Abstract View Model for Semantic Web', *1st International IFIP WG 12.5 Working Conference on Industrial Applications of Semantic Web (IASW '05)*, Springer IFIP Book Series, Jyvaskyla, Finland, pp. 227-46.

13. Rajugan, R., Chang, E, Feng, L & Dillon, TS 2005, 'Modeling Views for Semantic Web', *First IFIP WG 2.12 & WG 12.4 International Workshop on Web Semantics (SWWS '05), In conjunction with On The Move Federated Conferences (OTM '05)*, Springer-Verlag, Agia Napa, Cyprus, pp. 936-46.

14. Rajugan, R., Chang, E, Feng, L & Dillon, TS 2005, 'Semantic Modelling of e-Solutions Using a View Formalism with Conceptual & Logical Extensions', *3rd International IEEE Conference on Industrial Informatics (INDIN '05)*, IEEE Computer Society, Perth, Australia, pp. 286-93.

15. Rajugan, R., Chang, E, Dillon, TS & Feng, L 2005, 'Modeling Abstract Views as a Mechanism for Developing Sub-Ontologies', *The 1st IEEE International Conference on Next Generation Web Services Practices (NWeSP '05)*, IEEE Computer Society, Seoul, S.Korea.

16. Steele, R, Gardner, W, Rajugan, R. & Dillon, TS 2005, 'A Design Methodology for User Access Control (UAC) Middleware', *Proceedings of the IEEE International Conference on e-Technology, e-Commerce and e-Service (EEE '05)*, pp. 385-90.

17. Nassis, V, Rajugan, R., Dillon, TS & Rahayu, W 2005, 'A Systematic Design Approach for XML-View Driven Web Document Warehouses', *International Workshop on Ubiquitous Web Systems and Intelligence (UWSI '05), Colocated with ICCSA '05*, Singapore, pp. 914-24.

18. Rajugan, R., Chang, E & Dillon, TS 2005, 'Conceptual Design of an XML FACT Repository for Dispersed XML Document Warehouses & XML Marts', *The 5th International Conference on Computer and Information Technology (CIT '05)*, IEEE CS Press, Shanghai, China, pp. 141-9.

19. Rajugan, R., Chang, E, Dillon, TS & Feng, L 2005, 'XML Views, Part III: Modeling XML Conceptual Views Using UML', *7th International Conference on Enterprise Information Systems (ICEIS '05)*, INSTICC, Miami, USA, pp. 19-28.

20. Rajugan, R., Gardner, W, Chang, E & Dillon, TS 2005, 'xWeb: An XML View Based Web Engineering Methodology', *International Workshop on Ubiquitous Web Systems and Intelligence (UWSI '05), Colocated with ICCSA '05*, Singapore, pp. 1125-35.

21. **Rajugan, R., Chang, E & Dillon, TS 2005, 'Conceptual Design of an XML-View Driven, Global XML FACT Repository for XML Document Warehouses', *1st International Workshop on Data Management in Global Data Repositories (GRep '05), held in conjunction with the 16th International Conference on Database and Expert Systems Applications (DEXA '05)*, IEEE Computer Society, Copenhagen, Denmark, pp. 1139 - 44.

22. **Nassis, V, Rajugan, R., Dillon, TS & Rahayu, JW 2005, 'A Requirement Engineering Approach for Designing XML-View Driven, XML Document Warehouses', *The 29th Annual International Computer Software and Applications Conference (COMPSAC '05)*, IEEE Computer Society, Edinburgh, Scotland, pp. 388-95.

23. Rajugan, R., Chang, E, Feng, L & Dillon, TS 2004, 'XML Views, Part II: Modeling Conceptual Views Using XSemantic Nets', *Workshop & Special Session on Industrial Informatics, The 30th Annual Conference of the IEEE Industrial Electronics Society, IECON '04*, IEEE, S.Korea, pp. 454-62.

24. Gardner, W, Rajugan, R., Chang, E & Dillon, TS 2004, 'xPortal: XML View Based Web Portal Design', *17th International Conference on Software & Systems Engineering and their Applications (ICSSEA '04)*, Paris, France.

25. Nassis, V, Rajugan, R., Dillon, TS & Rahayu, W 2004, 'XML Document Warehouse Design', *6th International Conference on Data Warehousing and Knowledge Discovery (DaWaK '04)*, vol. 3181, ed. MKM Yahiko Kambayashi, Wolfram Wöß, Springer, Zaragoza, Spain, pp. 1-14.

26. **Rajugan, R., Chang, E, Dillon, TS & Feng, L 2003, 'XML Views: Part 1', *14th International Conference on Database and Expert Systems Applications (DEXA '03)*, vol. 2736, ed. WR Vladimír Marík, Olga Stepánková, Springer-Verlag, Prague, Czech Republic, pp. 148-59.

27. Chang, E, Dillon, T, Gardner, W, Talevski, A, Rajugan, R. & Kapnoullas, T 2003, 'A Virtual Logistics Network and an e-Hub as a Competitive Approach for Small to Medium Size Companies', *2nd International Human.Society@Internet Conference*, Springer-Verlag, Seoul, Korea, pp. 265-71.

28. Chang, E, Gardner, W, Talevski, A, Gautama, E, Rajugan, R., Kapnoullas, T & Satter, S 2001, 'Virtual Collaborative Logistics and B2B e-Commerce', *e-Business Conference*, Duxon Wellington, NZ.

Chapter 1

General Considerations and Motivation

1.1 Introduction

Electronic data authoring, publication and dissemination is a complex and challenging task. Only in the past decade, have publications of such information been made easier, mostly due to the success of the Internet and web technologies. Many people see published or distributed information as web pages, electronic documents, web (database) query results etc. Such web information, in comparison with processed structured data (such as in a relational database system) are usually unstructured, haphazardly organized, schema-less and exponentially increasing. In addition, there is a growing trend to produce and publish vast amounts of scientific and business data by automatically generating volumes of information by combining one or more distributed data and information sources, stored dispersedly over the Internet, for the consumption of both humans and machines (e.g. software modules, software agents etc.) In this context, it is important to provide database like features (Abiteboul, Buneman & Suciu 1999), such as representation, efficient querying and retrieving techniques, efficient storage and versioning, etc. in order for such large amounts of information to be of any use. Conversely, such large amounts of web information have generated enormous research interest and demand on traditional database systems and have created a new era of data models and technologies (Abiteboul et al. 1999; Abiteboul, Buneman & Suciu 1999; Abiteboul, Goldman et al. 1997; H Chan et al. 2001; S Chan, Dillon & Siu 2002; Do & Rahm 2004), such as (web) data warehouses, ontologies, autonomous information systems, middleware, etc.

In the new era of electronic data publication, meaningful information retrieval for a given context has proved to be a challenging task due the vast amount of unstructured and uncategorized data available to the users. The majority of this data is accumulated from the World Wide Web (WWW) (W3C-WWW 1997) and shared data repositories that are dispersedly stored over the Internet. The exponential growth of the web and its technologies, and the growing number of information sources gathered or stored at multiple locations, has created the problem of *standardization* vs. *customization*, where anyone can represent everything according to one's own representation and annotations. This triggered a series of new research directions such as metadata description languages, search engines (Luk et al. 2002), and knowledge representation (Aberer et al. 2004) techniques for the web to organize, describe, manage and disperse information in an orderly manner using the web as the new medium. To this end, many initiatives have been proposed, namely: (a) the formation of new standard bodies and consortiums, to define and promote standards for the web, such as the WWW consortium (W3C-WWW 1997), ebXML (www.ebxml.org), OASIS (http://www.oasis-open.org) etc., (b) new data models (HTML (W3C-HTML 1997), XML (W3C-XML 2004), XML Schema (W3C-XSD 2001), RDF (W3C-RDF 2004), OWL (W3C-OWL 2004)), etc., (c) transformation and query languages proposals (e.g. XQuery (W3C-XQuery 2004), RDQL (W3C-RDQL 2004), XSLT (W3C-XSL 2003)) and (d) new technological frameworks (e.g. Web Services (W3C-WS 2002)) and (e) new knowledge representation paradigms such as the Semantic Web (W3C-SW 2005).

Here, we next look at data models and data modelling as the first step in identifying problems and issues in designing applications for the new generation of data-intensive, web based application systems. Later in this thesis, we will carefully identify one such issue, namely *views*, and explore the issues of designing data intensive applications using views.

1.2 Data Models

Data models are formulated to abstract, describe and constrain the domain that the model is intending to model or design. The intention is to model the domain so that it provides the foundation that can produce a constant flow of correct, complete and consistent information for all stakeholders and communities of the domain in which they are involved; producing, processing and consuming the information. In this context, a good data model provides a descriptive *schema*, as the basis to describe and constrain the possible information of a domain. In order to produce such a descriptive schema model which captures all the required structures and properties of a domain, there exists a prerequisite that there be a complete and clear understanding of both (a) domain in question, and (b) the nature of the information that needs to be modeled.

1.2.1 Data Models and Schema

In traditional data modelling, the emphasis is always on producing the best descriptive schema in as much detail as possible, given the domain and its information requirements. Also, a data schema is produced *before* the creation of the data itself (Florescu 2005). In producing such schemas, it is always assumed that there exists, unbound, a prior knowledge of the domain, its information and its properties. This process worked well for many solutions such as enterprise databases, software applications and textual databases, where the information produced, processed and consumed is predictable and has had uniform definitions and descriptions. Thus, many successful data modelling techniques, from Entity-Relationship Diagrams/Data-Flow-Diagrams (ER-D/DFD) to highly successful Object-Oriented conceptual modelling (OOCM), were developed and refined to cater for different domains and paradigms. In addition, data (representation) models, such as network, hierarchical, the commercially successful relational, OO models were developed and are described (and constrained) by the predefined domain specific schemas. Since these models were based on, and produced, uniform schemas of a domain, and the allowable data structures and formats were predefined and constrained by the schemas, the data models are said to be handling *structured data*. However, in some domains, due to the nature of the domains and other limitations, a prior knowledge of the structure of the information that needs to be processed, produced and consumed is never available until particular segments of such information are physically available (Florescu 2005). Thus, producing a schema using the traditional data models failed in such domains and many work-around solutions were developed to model, design and deploy such domains. Such data domains are frequently referred to as dealing with *unstructured* data.

1.2.2 The "Web" Data Model

The unexploited domain of unstructured data gained momentum with the introduction of the World Wide Web (WWW) and the Internet during the late '80s and early '90s. With the extreme simplicity of electronic authoring (namely using HyperText Mark Language (HTML) (W3C-HTML 1997)) and as the speed and efficiency of disseminating such authored documents increased, managing, archiving and processing such vast amounts of documents for information, without a predefined structure and/or schema (i.e. unstructured data), provided a challenge to traditional data modelers. The commercially successful relational model, which is capable of managing and processing large amounts of data for information and ad hoc queries, could not address the challenge of web data. However, with limited commercial success, the OO data models provided some limited success in modelling unstructured data (mostly by forcing some form of incomplete schema structures onto the web data) but could not provide sufficient support for representing and describing web data.

The problem of modelling and managing of web data, combined with the shortcomings of traditional data models in handling unstructured data, created a new breed of data models, namely *semi-structured* data models. Semi-structured data models contain data that are often *self-describing* and *schemaless* (Abiteboul, Buneman & Suciu 1999). As stated earlier, in comparison with traditional (structured) data, where the data structures and data types are described first in a schema, in semi-structured data, the structures and types are described using a simple syntax (thus schemaless) and usually embedded within the data instances (thus self-describing). Such data are usually dominant in electronic document publishing, web publishing and many e-Science disciplines such as geological databases, gene molecular biology (GO /GONG, http://gong.man.ac.uk), health sciences (Hadzic & Chang 2004), Medical Information Systems (Wollersheim & Rahayu 2005) astronomy (AstroGrid), aircraft engineering (Geodise) and many legacy information systems (e.g. in legal, financial, supply chain industries and government institutions).

1.2.3 The Semi-Structured Data Model & XML

The concept of semi-structured (and unstructured) data models that were previously confined mainly to certain data domains (e.g. web), does not hold true today. Today, there are many initiatives and forums that are involved in active research, development and deployment of adapting a semi-structured data model to traditional and non-web based domains to utilize and harness the power and flexibility of semi-structured data models to software applications. Thus, such a universal appeal for semi-structured data model creates a need to investigate modelling, representing and retrieving techniques and solutions for semi-structured data models. In this area, XML has increasingly taken up a significant role.

The eXtensible Markup Language (XML) is emerging as the core data representation (and storage structure) standard for the web, from annotated web pages to Semantic Web and ontology bases. Since its adoption by the W3C as a recommendation in 1996, which was derived from its predecessor SGML, XML was originally intended to be a simple, yet meaningful data description and encoding format for electronic data representation, publishing and dissemination over the web, readable by both human and machines. Its tag-based, self-descriptive, *semi-structured* data model provides facilities for user-defined tags and description that makes authoring of XML documents easier and customizable, yet allows external readers (humans, agents and applications) to interpret or integrate them correctly. Due to this, the core data model for the majority of the new and emerging web technologies, such as XML Schema, XQueryX, Web Databases, Web Services, Semantic Web, web protocols (e.g. SOAP, XAML) are based on an XML document structure.

XML, from a data modelling point of view, can be considered as *semi-structured*. From a database point of view, it can be considered as a document rather than an inter-connected collection of data elements. Since it is self-descriptive, tag descriptions are embedded within the document, thus enabling the reader (human or machine) to interpret the content without a need for managed metadata repositories or metadata schema. But, an XML document by itself cannot provide validity to a document (i.e. does the document is lexically correct as the author intended?). For this purpose, the W3C proposed Document-Type-Definition (DTD) (W3C-DTD 2001) and its successor XML Schema (XSD) (W3C-XSD 2001)are used to describe and constrain XML documents.

DTD (and XSD) contain user-defined element type definitions and attribute descriptions to describe an XML document, its structure and the meaning of its content. The replacement of DTD is the XML Schema, which is richer in vocabularies and considerably extends the capabilities of DTD for defining and constraining the content of XML documents (Feng, Chang & Dillon 2002; Feng, Chang & Dillon 2003). The XSD, which itself is represented in XML, provides a set of advanced constructs, such as XML document usage, meaning, relationships and a rich set of constraints, that can be grouped into three groups of twelve components, to define and constraint XML documents. One of the main features of the XML/XML Schema is the separation of document content from its presentation and structure.

In simple terms, XML represents a hierarchical structure of repeated fields and values with embedded structural information in the form of a tag. This represents an ideal data model for semi-structured (and unstructured) data that can be used for efficiently representing and retrieving information in an orderly manner, similar to traditional relational and Object-Oriented (OO) models. But, unlike successful relational (and OO) models, XML lacks modelling and querying techniques, a problem that is addressed in the following section.

1.3 Data Modelling & Challenges

Data modelling is a term used by database theorists to describe how a domain is represented conceptually and logically, at the needed level of abstraction of the required information, using a given data model. Since the early days of data modelling, many techniques have been developed from time to time to improve and refine the engineering of data to overcome new and emerging challenges and changing data requirements (i.e. data formats, data models, database systems etc.). These include early Jackson methodology to ERD and later Object-Oriented (OO) conceptual modelling (Dillon & Tan 1993; Graham, Wills & O'Callaghan 2001), as shown in Fig. 1.1. To date, the OO modelling technique is one of the foremost and highly successful data modelling paradigms developed that is capable of modelling real-world objects, relationships, constraints, behaviour and constructs, focusing on semantic, structural and dynamic aspects at varying level of abstraction, that is precise and comprehensible to the users (Dillon & Tan 1993).

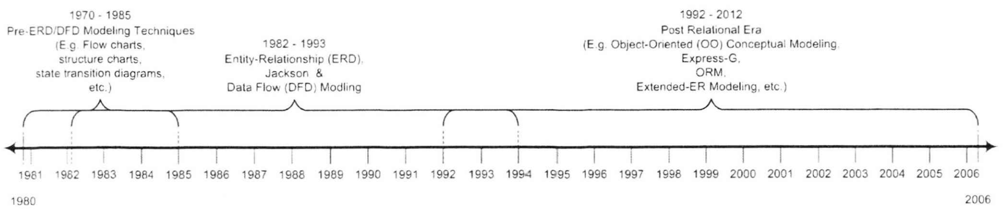


Figure 1.1: Data modelling methodologies

In practice, an OO model provides the blueprint for a system or problem being considered. From a data modelling point of view, OO has been successfully applied to solve complex problems such as real-time databases to data warehousing and OLAP queries. The success of the OO model is mainly due to its ability to capture, model and communicate the problem in question with the required level of abstraction using industry standard notations and techniques.

Similarly, for structured data (noticeably relational data), the database technologies have proven their success in many areas, such as models, efficient storage and retrieval techniques, performance improvement techniques, model automation and transformation tool etc. Unlike semi-structured data (i.e. XML) models, for the structured data, data storage, retrieval and manipulation techniques have evolved to provide the performance and Return On Investment (ROI), that is acceptable to industry and enterprise standards. Thus, it is interesting to investigate OO and traditional database concepts, theory and practice approaches, in the context of web and XML.

In the context of data modelling, though XML is rich in data descriptions and unambiguous structural representation, XML focuses on syntactic structure rather than data semantics and their relationships (Feng, Chang & Dillon 2002; Feng, Chang & Dillon 2003). In combination with XML Schema, which provides rich facilities for constraining and defining XML content, it still lacks the ability to provide higher levels of abstraction (such as conceptual models) that are easily understood by humans (Feng, Chang & Dillon 2002; Feng, Chang & Dillon 2003). Conversely, OO modelling languages and techniques provide insufficient modelling constructs for utilizing data descriptions, relationships and constraints that are unique to semi-structured data models (e.g. XML) and schemas (e.g. XML Schema). Thus modelling and representing real-world objects using XML/XML Schema still proves to be a challenging task. The next section addresses some of the challenges faced in developing data models for XML.

Engineering successful semi-structured data solutions (e.g. XML data warehousing) involves not one, but many, technologies and techniques from many different fields (Florescu 2005). These technologies and techniques are adopted or originated from disciplines such as, data modelling, knowledge representation, information retrieval, data storage and indexing techniques, authoring etc.

It is imperative that one should look at some of the present and emerging challenges faced by XML, as being a new data model for today's and future applications. Most of these challenges are due to the lack of proven data modelling techniques for semi-structured data models, namely XML, such as, view models, storage model, query models and language standards, etc. while some of the other challenges are due to the new and emerging software engineering initiatives such as Model Driven Architecture (MDA) (OMG-MDA 2003). Firstly, we look at some of the drawbacks of XML due to lack of supporting database techniques.

XML, which is a robust data model for multi-format data, should also be able to support emerging technologies and modelling techniques such as MDA. The success of the relational model is its adaptability to new and upcoming modelling techniques that were proposed during its evolution. Similarly, XML should be able to adapt to, not just new data standards, but also new modelling techniques. At the moment, MDA is one such modelling technique, which has support from various industry standard bodies for developing successful software solutions. Thus, it is interesting to investigate engineering XML solutions under the MDA initiates.

The introduction of MDA initiative by OMG, platform independent models play a vital role in system development and data modelling. Under the MDA initiative, first the model of a system is specified via an abstraction notation that is independent of the technical or deployment specifications (i.e. Platform Independent Model or PIM) and then the PIM is mapped or transformed into a deployment model (i.e. Platform Specific Model or PSM) by adding platform or deployment specific information. To support MDA initiatives in data modelling, data semantics, constraints and model requirements have to be specified precisely at a higher level of abstraction.

This presents an opportunity to investigate data views as a means of providing data abstraction and semantics in PIMs for data intensive, MDA solutions, such as XML document warehouses. But, the existing OO paradigm and modelling languages provide minimal or no semantics to capture abstract view formalisms (at the conceptual and logical levels), and existing semi-structured data technology standards and languages do not provide support for concrete view formalisms.

In the context of MDA solutions for XML domains, it is still a challenging task to produce PIMs despite the flexibility and the semantic richness of the semi-structured schema languages. This is mainly due to OO modelling languages such as OMG UML™ (OMG-UML™ 2003), Extended-ER (Elmasri & Navathe 2004) etc. provide insufficient modelling constructs for utilizing XML schema like data descriptions and constraints, while XML Schema lacks the ability to provide higher levels of abstraction (such as conceptual models, visual constraints, etc.) that are easily understood by humans. This is in conformance with the notion that models are often abstract representations which keep only as much of the detail as is relevant to the particular problem being considered (Dillon & Tan 1993; Graham, Wills & O'Callaghan 2001). In this context, XML Schema generally is too low a representation to permit users to interact, visualize or understand it. To rectify this situation, many researchers in works such as (Chen, Ling & Lee 2002; Conrad, Scheffner & Freytag 2000; Feng, Chang & Dillon 2002; Feng, Chang & Dillon 2003), have applied intuitive techniques, notations and transformation methodologies to capture XML semantics at the conceptual level.

1.4 The View

The notion of a *view* is an important database concept that facilitates *stored* data to be seen from different view points without reorganization. Naturally, the original intended motivations for views are to increase flexibility and user access control for the data stored in database systems. From a data modelling perspective, view models may be considered as a mechanism that provides some form of data "abstraction" and "user perspective" to the underlying stored data in a database. In the context of database administration, views provide flexible yet inexpensive user access control mechanisms and persistent query facilities for frequently used complex queries. From a user perspective, views provide shorthand for frequent user queries and grouping of arbitrary portions of stored, user-accessible data into one or logical units of data, that make sense to him/her. From a programming point of view, views provide excellent Application Programming Interfaces to the stored data, without directly accessing each individual data store. One of the important characteristics of the view model is that view definitions and (view) data access are able to be incorporated into any database applications, programming interfaces and data access modules, without major changes to the application interface, database schema, data and the storage I/O structure of the interfaces.

1.4.1 View Properties

A view has been defined as a query expression specified using a particular query language, and the view construction is usually handled by the data model specific database management system (Codd 1990; Date 2003; Elmasri & Navathe 2004). However, views are represented only by their names and definitions in almost all of the view models. Unlike search queries (e.g. keyword search on the web), database queries are not of a procedural nature being formulated using the stored, well-defined meta-data semantics and schemas. Therefore, a result of a view query expression is a well-defined data set that conforms to the original meta-data schemas. In addition, view definitions do not require knowledge of the storage representation, paths or I/O access methods. Thus, even today, a view is still considered to be a *stored* (or saved) query that returns certain information from the stored data source on request (by user or application). In summary, one can stereotype the existing view formalisms using the following properties.

(i) Views are virtual entities (table, class etc.) represented only by their names and definitions using a database query language. There exists no explicit distinction between a view and a stored entity for the purpose of user (or user-application) queries and data retrieval request (Codd 1990), except for data model restrictions.

(ii) View definitions may be expressed using stored entities (tables, classes etc.), other view definitions, or the combination of both stored entities and other view definitions. Such view definitions are usually not procedural in nature and do not contain explicit database or storage specific information. Also, view definition should return results for queries issued against it, usually using same language as the view definitions (e.g. SQL).

(iii) View specification (supported by the view specification language/(s)) should allow a mechanism to import arbitrary source database objects (or a section of those objects) into one view object. Also, it should support the creation of new database objects to be included in the view definition.

(iv) Views should be able to have their own hierarchy similar to that of the stored data entities. A view hierarchy should be allowed to intermix with the hierarchy of stored data entities if, and only if, it does not disrupt stored data entities. Usually, view hierarchies and the hierarchies of stored data entities should be kept separate and independent of each other (Elizabeth Chang 1996; Kim & Kelly 1995).

(v) Views should able to be materialized on request and stored for faster access, performance or as a data source for other view definitions, if necessary for a specified time. The view specification language should provide some mechanism to maintain data integrity and consistency, i.e. being responsive to the state changes of stored data sources, during the lifetime of a materialized view.

(vi) View updates are allowed if, and only if, they do not violate the semantics of the stored data entities.

(vii) The data domain of views is not explicitly specified during view definition. Usually, it is bound at run-time to the domain of the union of stored data entity/(ies), from which the view is being populated. In the case of new derived data types, the view specification language should allow mechanisms for such data types to be defined and instantiated at view run-time.

1.4.2 Views and Data Models

Since the early days, view models have been studied extensively in the context of: (a) relational data models (Codd 1990; Date 2003; Elmasri & Navathe 2004; Gupta, Mumick & (eds) 1999), (b) object-oriented data models (Abiteboul & Bonner 1991; Bertino 1992; Date & Darwen 1998; Kim 1990; Kim & Kelly 1995) (c) graph-structured data models (Zhuge & Garcia-Molina 1998) and most recently the semi-structured data models (Abiteboul 1999; Abiteboul et al. 1999; Abiteboul, Goldman et al. 1997; Chen, Ling & Lee 2002; Volz, Oberle & Studer 2003; Wouters et al. 2004b).

In relational databases, views (Codd 1990) are part of the external schema of the ANSI three-schema (i.e. conceptual, storage and external) architecture (Date 2003; Kim & Kelly 1995), where persistent view definitions are instantiated when a query is issued against the view. Conversely, new view definitions can be created and/or deleted without affecting the consistency, structure and storage space of the overall database schemas and data. In theory, a relational view is considered to be a collection of *virtual tuples* that form a *virtual relation* (or table) (Date 2003; Elmasri & Navathe 2004). Also, the view definitions are expressed in terms of (Codd 1990): (a) base or stored relation/(s), (b) other virtual relations (i.e. other views) and (c) a combination of (a) and (b). Usually, relational views are defined using the SQL '92 (Date 2003; Elmasri & Navathe 2004) compatible relational query languages and the view construction, materialization and updates are managed by the Database Management System (DBMS). Since the early conventional relational data model was restricted to only base data types, the presumed usage for views was to provide data protection against unauthorized user access. Later, when relational data models were extended to support complex data types and functions, the concept of views was extended to support selective updates (i.e. using view updates), data partitioning (vertical and horizontal), data integration, complex and aggregate/summary queries (such as data warehouse queries).

Due to its perceived benefits, during the OO revolution, the core concept of views was extended to OO and Object-Relational (O-R) data models. A class in an OO system has both attributes (may be nested or set valued) and methods. A class can also form complex hierarchies (inheritance, part-of, etc.) with other classes. Some of the early discussions on OO view models contributed to the notion of an extended relational model (Kim 1990). Naturally, this statement reflected on many early discussions of OO views (Abiteboul & Bonner 1991; Kim 1990; Kim & Kelly 1995). Thus, in OO systems, the views were defined in a synonymous manner to the relational model and/or extending the relational definition. They included the notion of *virtual classes*. But, unlike in the relational view model, where logically a virtual relation is just another relation, as stated earlier, in the OO model, in addition to being a virtual class, a view should be placed within a scope of a class hierarchy that is reflective and responsive to system and schema changes. In simple terms, an OO view mechanism should be able to provide a user with a class hierarchy that is more appropriate to one's (or applications') needs rather than the actual stored hierarchy of the domain in question. Some classes may be hidden and new classes may be introduced. This issue is one of the unique characteristics of the OO view model and has not been resolved completely in many commercial and/or the research literature to this date. Thus, due to these complexities, the definition of views for OO systems is not straightforward and complete.

Further, both relational and OO view concepts make two implicit assumptions, that the underlying data is structured and there exists a fixed data model and a data access/query language. In the relational model, SQL is the core language of choice for view definition. The SQL standard provides view specification and definition as part of the language constructs. However, the notion of query languages including view specification and definition in the OO data model is not universally agreed upon and standardized. Most popular OO query languages support view construction in a restrictive manner that the characteristics are comparable to relational views rather than OO views.

On the other hand, since the introduction of the relational data model (Codd 1983; Date 2003; Elmasri & Navathe 2004), motivation for views has changed over the last two decades. At present, views are widely used in:

(i) user access and user access control applications

(ii) defining user perspectives/profiles (Elizabeth Chang & Dillon 1994)

(iii) designing data perspectives

(iv) dimensional data modelling (Gopalkrishnan, Li & Karlapalem 1999; Mohania, Karlapalem & Kambayashi 1999; Rafanelli & (ed) 2003)

(v) providing improved performance and logical abstraction (materialized views) in data warehouse/OLAP and web-data cache environments (Gopalkrishnan, Li & Karlapalem 1999; Gupta, Mumick & (eds) 1999; Mohania, Karlapalem & Kambayashi 1999; Rafanelli & (ed) 2003)

(vi) web portals & profiles

(vii) extraction of document structures or sub graphs in semi-structured documents (Abiteboul, Goldman et al. 1997)

(viii) Semantic Web, for sub-ontology or Ontology views (Volz, Oberle & Studer 2003; Wouters et al. 2004a).

From this list, it is apparent that the uses and applications of views are considerably beyond that initially envisioned by Codd et al. (Codd 1990) (i.e., data elaboration vs. the 2-Es; data Extraction and Elaboration), with extensive research being carried out by both researchers and industry to improve view design, construction, and performance.

However, with the introduction of web data and *semi-structured* data models, the motivation for a view model to deal with semi-structured data resulted in many intuitive approaches being proposed. Naturally, many research directions opted to adopt existing view models, mainly OO (Abiteboul, Goldman et al. 1997; Abiteboul, Quass, McHugh & Widom 1997; Abiteboul, Quass, McHugh, Widom et al. 1997) and graph based models (Zhuge & Garcia-Molina 1998), for the semi-structured data paradigm.

1.5 Motivation

As stated in the previous sections, view models provide major benefits in data modelling. In its current form, views are: (a) easy to define, (b) simple to derive, (c) independent of data storage structure, and (d) independent of the programming languages. This characteristics of views helps to design data intensive architectural constructors such as data cubes (Humphris, Hawkins & Dy 1999; Ponniah 2001), operational data stores (Inmon, Imhoff & Battas 1996; Kimball & Ross 2002), and persistent data interfaces etc.

Our first motivation for this work is the idea of view concept vs. view construction syntax. Contrary to popular belief, even with traditional data models such as relational or OO, no formal semantics exist for views. To this date, a view is simply defined as a query expression (specified using a query language syntax), that is equivalent of a virtual table (in relational model) or virtual class (in OO models). However, there exist many algorithmic solutions (including refinement, maintenance and tuning) for constructing and maintaining views in many data models (including XML) using one or more query languages. But, in regards to semi-structured data and XML, there exist no widely accepted standards, recommendations or language syntaxes to support view construction. Thus, it is still a challenging task to define or use views, where, in the case of needing to utilize the concept of views in a data model (or query language) that does not support view construction and/or provide necessary and widely accepted language syntax for defining views. Therefore, it is interesting to look at formally defining the notion of a view, similar to that of other concepts such as the notion of variables, list, set or arrays, that is adaptable to any data model (e.g. XML) or query language.

Most semi-structured data and XML, in its native form is usually a textual representation of the data they represent. It is not refined, optimized or normalized for storage or query processing in its native form. Another unique feature of semi-structured data (including XML) is the document size, structural complexities and their growth rate, due to embedded semantics (both synthetic and structural). Therefore, data access and manipulation is a computationally expensive iterative task, in comparison to accessing a tuple in the semantically flat relational model. From past experience, in the history of data models, the success of a data model heavily depends on its query processing features and the performance issues associated with it. Thus, one of the major challenges for the XML data model is the capability of having native yet optimized storage and query processing features that are comparable to the performance of the relational model. In many related works (Abiteboul, Quass, McHugh & Widom 1997; Braga, Campi & Ceri 2005; Ceri et al. 1999; Lucie-Xyleme 2001; XMark 2003), these issues are being addressed, from a straightforward XML DBMS context to web scale XML query processing. But, in addressing these issues (which is outside the scope of this thesis), many compromises have to made in regards to the XML data model, from loss of data semantics (to add improvements to its storage structure, for e.g. XML-enabled relational DBMS) to the collapse of document structures (to increase query and/or data manipulation performance, for e.g. XML data access via complex indexing algorithms which compresses or flattens the hierarchical XML document structures).

Thus, the second motivation for our work is the idea of using views to provide sub-sections (or sub-trees) of XML document trees that is of interest to the user, thus providing the user (and user application) with a simplified mini-document of the original, larger document. This will reduce complex iterative query processing and reduce computationally expensive data manipulation operations as the users (and user applications) are given only the portion of the larger document that is of interest to them, at a given point in time.

Another important property of XML (and common to most semi-structured data models) is its ordered structure. Unlike in relational (and commonly in OO) data model, where a simple joining of tables produces a result set that is independent of column ordering. That is, in the result, changing the position of a column have no consequence to the data semantics (or the meaning) of the result set. But in XML, joining two or more document clusters needs to consider original ordering of the document clusters as, depending on the depth the XML document structure many results are possible, each result being different in structure and semantics, corresponding to new XML data (and document structure). This is because, unlike relational data, XML is tree-structured (Feng & Dillon 2005; Feng et al. 2003) and order is one of the most common construct in a tree-structured (or graph structured) item. In data intensive applications such as (XML) data mining (Feng & Dillon 2005; Tan et al. 2005) and (XML) data warehousing ordering is one of the most important and challenging property to deal with.

Thus, the third motivation for our work is similar to our second, where we use views to provide meaningful structures that are of use to the users. This is because an XML document structure is complex and large in nature and requires complex validation and computationally expensive algorithms to maintain them in their entirety. Also, in comparison to query expressions and extraction algorithms for semi-structured data, a view mechanism is capable of providing meaningful data extraction and structure to an XML document. Also, such a view needs to be used as contexts to interpret further query issued against its definition and/or structure. Therefore, in theory, an XML view will behave as a mini-document database with preserved structural integrity and semantics of the parent document, thus users (and applications) use only a portion (or sub-trees) of the logically grouped document and its structure resulting in improved performance and loss of complexities. Furthermore, this introduces the requirement that view is a valid view.

One of the feature of XML (and the semi-structured data in general) that contrasts with the structured data domain is the heterogeneity within a document (and the associated documents). Therefore, when adapting traditional database concepts like views to XML, one needs to address this issue carefully. In semi-structured data paradigms, this issue is easily dealt with by adopting a self-describing encoding notation and providing some data definition and manipulation features at a higher-level than the traditional query language expression, as a simple query syntax cannot deal with complex iterative operations that require access to one or more schema and data domains. However, as we have repeatedly stated in this chapter, a view is usually treated as a query expression in almost all data models known to us today. Also, it has been demonstrated in relational and OO models that, views are capable of being used in architectural constructs (Cluet, Veltri & Vodislav 2001; Gopalkrishnan, Li &

Karlapalem 1999; Gupta, Mumick & (eds) 1999; Inmon, Imhoff & Battas 1996; Jeong & Hsu 2001; Mohania, Karlapalem & Kambayashi 1999) such as data interfaces, data cubes, data wrappers, dimensional hierarchies, etc. To construct similar models in semi-structured, using existing view concepts, there is the need for a highly expressive (similar to that of OO programming languages with complex data constructs) and flexible query language to define such architectural constructs. But since query languages are lower level constructs than conceptual representations, the availability of views is also restricted to low-levels abstraction, thus constraining the architectural constructs to be designed only at the language level. Thus, this also gives rise to the idea of conceptualizing the notion of views.

Therefore, one of the strongest motivations for this work yet, is the idea of looking at views from a data modelling point of view, in the context of XML, for designing and deploying architectural constructs and database applications. To achieve such a challenging task, we need to look at semi-structured (namely XML) views from an unorthodox perspective, namely: (a) views need to be first-class citizens in the data model with semantics and properties just like regular stored objects; (b) provide systematic construction (and transformation) of views as in the case of stored domain objects; (b) the view must provide structures that are rich enough to represent the complex semantics, (c) views should provide flexible customization mechanisms; and (d) views should provide extendable properties that are independent of query languages and data models. In simple terms, it is an intuitive approach to investigate expressing views at a higher-level of abstraction than at the query language level.

Since the early days, conceptualization of programming constructs has led to valuable concepts being modeled and designed, at a higher level of abstraction in a precise and comprehensible manner. A good example of this is the conceptualization of the OO programming language concepts such as classes, composition, inheritance etc. which led to one of the effective ways of solving real-world software problems: the widely adapted OO conceptual modelling paradigm. But, since the early relational models, despite usefulness of views, no work has been devoted to addressing the issue of conceptualizing views, thereby providing view definitions and specification at a higher level of abstraction than at the query language level. To the best of our knowledge, this is one of the only researches that looks at conceptualizing the notion of views in the context of semi-structured data (i.e. XML and SW).

1.6 Aim & Scope of Thesis

In software development, particularly in data modelling, the development process is normally divided into three stages, namely *analysis*, *design*, and *implementation*. In the *analysis* stage the problem domain is analysed by using a particular data modelling technique, such as object-oriented modelling. The richness of object-oriented modelling has been widely acknowledged. The output of this stage includes platform independent models, conceptual semantic constraint definitions and data model definitions.

In the design stage, the results from the previous stage are translated into an implementation-ready format, such as design models, schemas and software structure models. In the implementation stage, the conceptual models are implemented in platform specific executable modules. Such modules can be domain specific, language specific or both. The scope of this thesis focuses mainly on the first two stages and, for the implementation stage, provides domain specific architectural constructs and/or models.

The main aim of this thesis is to provide a theoretical and methodological framework for the concept of view, with conceptual and schemata extensions, in the context of semi-structured data (namely XML). Also, the view model is accompanied by a systematic view design methodology. Furthermore, as a proof of concept, the proposed view model is applied to three data intensive domains, where architectural constructs are designed using the views. This is in addition to two illustrative case studies (one data-centric and the other document-centric), where an illustrative practical walkthrough is given to demonstrate the unique features of the view model.

In summary, in this thesis, we look at XML as a data model for engineering data intensive solutions. For such a challenging task, first we look at data abstraction in the form of extension at the conceptual, logical and instances levels, in the context of views. Secondly, we look at these views from the data modelling context, which is to provide modelling and design constructs at the varying levels of abstraction. Finally, we look at engineering some XML applications at different levels of abstraction, such as the case in MDA. It should be noted that the intention of this thesis is not to propose a new view standard for XML or extensions to XML query languages.

As we have indicated so far, the concepts presented in this thesis adhere to the aims and scope as described above. But, due to reasons of time availability and some technological restrictions, some limitations to the scope of the thesis, have to be set. The issues that fall outside the scope of this thesis are:

(i) Taking a standpoint on the best storage structure for XML: throughout this thesis, we do not claim a particular storage structure is better for XML or XML query processing.

(ii) Viewing extensions: this thesis considers only the extraction and elaboration features of the view model addressed, and not extensions, as described in (Wouters 2006).

(iii) Formal query model: there exist numerous query languages and query models for querying XML documents. This intention of this thesis is to propose neither a new query language nor query language extensions to existing query languages.

(iv) Query tuning and optimization: query tuning and optimization is a serious but necessary process in improving query performances. Here, we do not address particular query tuning or optimization techniques or propose new techniques. The view query expressions are in their original form without tuning or optimization, using straightforward query engines, querying DBMS-independent XML documents.

(v) A view implementation mechanism such as that given in (Abiteboul, Goldman et al. 1997; Aguilera et al. 2002; Lucie-Xyleme 2001; Xyleme 2001).

1.7 Plan of the Thesis

In this thesis, we adopt the "*system development as an IS research methodology*" (or proof of concept by system development) (Nunamaker, Chen & Purdin 1991) as the research methodology. It is one of the important research methodologies for research and development projects in Information Systems. In adopting such a research method, we investigate not just the effectiveness of our concept, but are also able to demonstrate how our concepts actually work in real-world scenarios.

This thesis is organized into five parts (Fig. 1.2) spanning eleven chapters. The dependency and inter-relationships between the chapters are depicted in Fig 1.3.

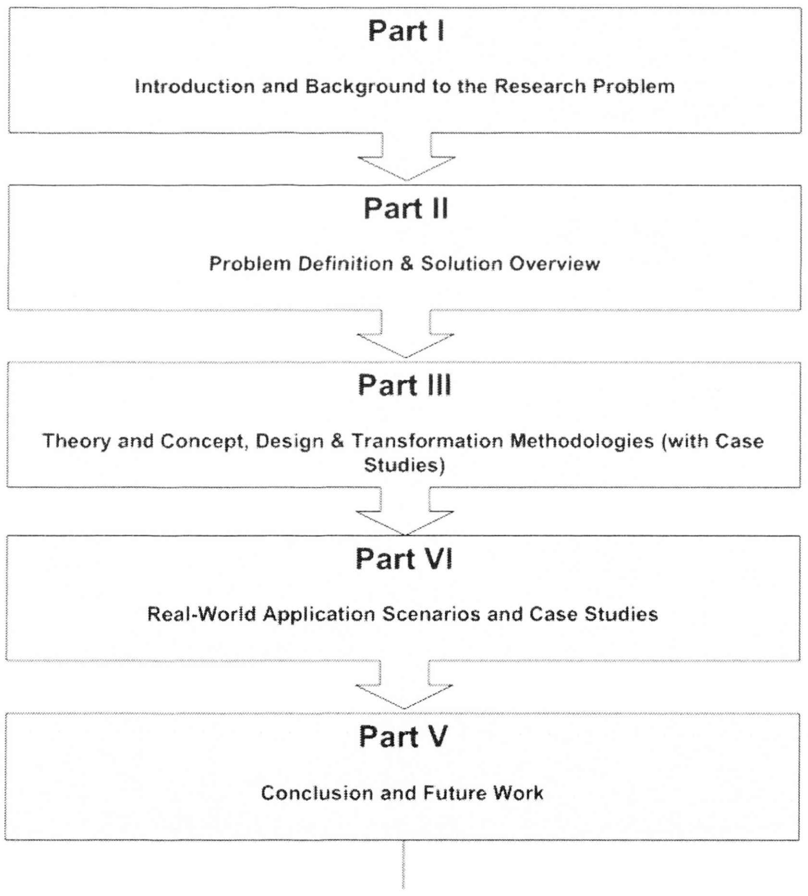

Part I
Introduction and Background to the Research Problem
Part II
Problem Definition & Solution Overview
Part III
Theory and Concept, Design & Transformation Methodologies (with Case Studies)
Part VI
Real-World Application Scenarios and Case Studies
Part V
Conclusion and Future Work

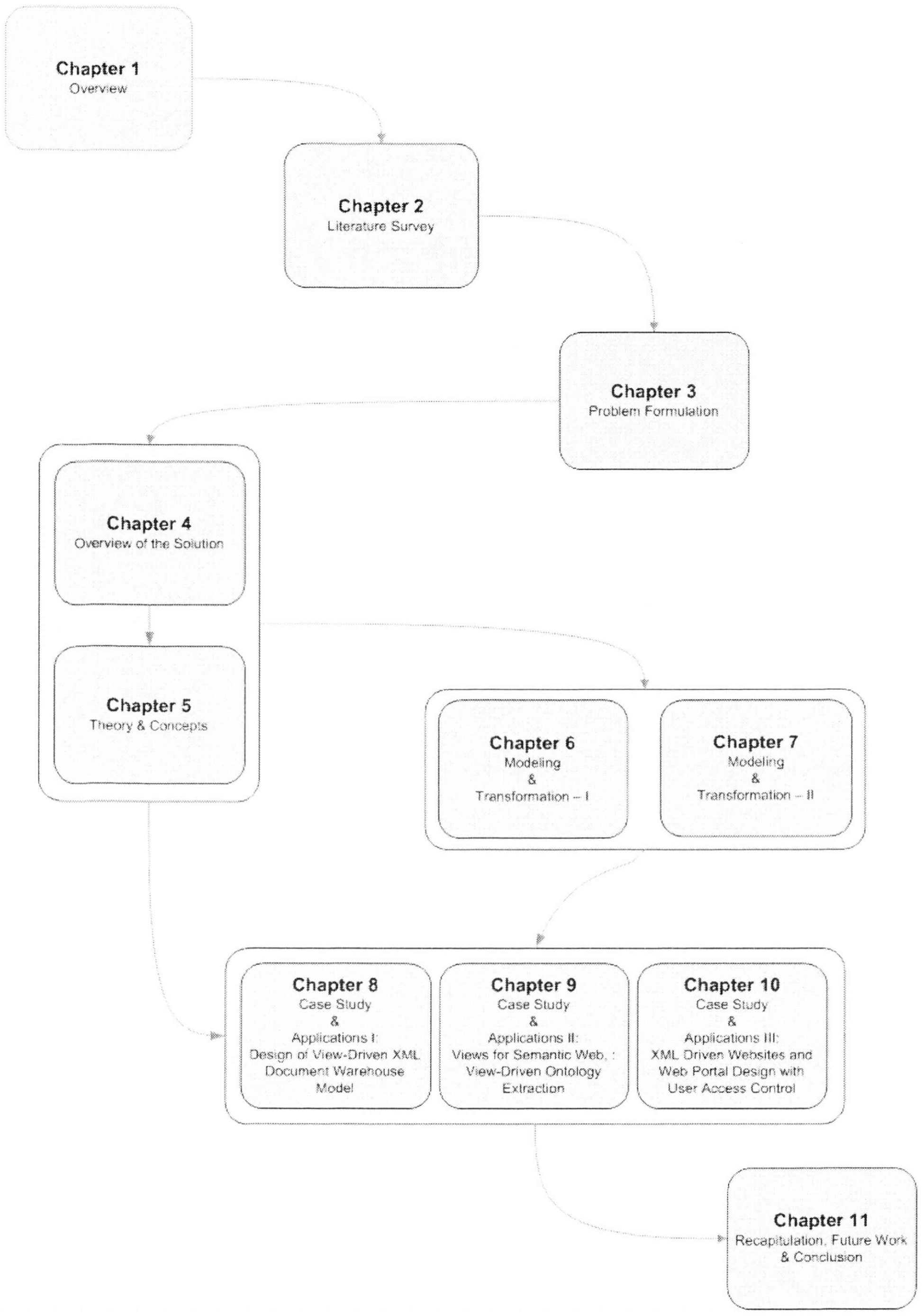
Figure 1.2: Grouping of chapters

Chapter 2 discusses existing conventional and new view models. These view models may be grouped into four major categories based on data models for which they are designed for, namely: (i) classical (or relational) views, (ii) Object-Oriented (OO) views, (iii) semi-structured (namely XML) view models and (iv) view models for Semantic Web (W3C-SW 2005). The motivation of this chapter is to outline and carefully analyse existing work done in view models over recent years and highlight problems that remain outstanding.

Chapter 3 describes in detail the problem addressed in this thesis. First, it provides some background information on some of the concepts, definitions, technologies and terminologies that are imperative to understand the problem addressed in this research followed by a formal statement of the problem.

Chapter 4 presents the overview of the solution this thesis intends to propose. Here, first an overview of the notion of XML views, view concepts, properties and the XML view terminologies are presented. This is followed by a brief discussion on modelling and transformation issues related to the XML views are described. Another unique XML specific view property, the view hierarchy and how it is dealt with in the XML view model, is also discussed. This chapter includes a detailed description of two example case studies used in this research to illustrate the theory and concepts, namely: (a) a simple conference publishing system (CPS) similar to that of systems such as IEEE Xplore (IEEE 2004) or Springer (Springer 2005) conference publishing systems, and (b) an e-Logistics system for global cold storage, warehousing and logistics booking system (e-Sol) (ITEC 2002).

The rest of the thesis is divided into three main parts (Fig. 1.2): (i) the semantics, concepts and the definitions of XML views; (ii) modelling and transformation of XML views; and (iii) application of the XML views in real-world scenarios. The first part is addressed in Chapter 5, followed by the second part in Chapters 6 and 7, as a practical walkthrough of the modelling issues and the transformation methodologies associated with XML views using two different modelling languages. The third part is presented in Chapters 8, 9 and 10 (Fig. 1.3).



Chapter 1
Overview
Chapter 2
Literature Survey
Chapter 3
Problem Formulation
Chapter 4
Overview of the Solution
Chapter 5
Theory & Concepts
Chapter 6
Modeling & Transformation – I
Chapter 7
Modeling & Transformation – II
Chapter 8
Case Study & Applications I: Design of View-Driven XML Document Warehouse Model
Chapter 9
Case Study & Applications II: Views for Semantic Web, : View-Driven Ontology Extraction
Chapter 10
Case Study & Applications III: XML Driven Websites and Web Portal Design with User Access Control
Chapter 11
Recapitulation, Future Work & Conclusion

Figure 1.3: Plan of chapters and their inter-relationships

Chapter 5 (titled Theory & Concept) elaborates further on Chapter 4 and provides a formal definition of the XML views, including formal properties, semantics and characteristics of the view model. To the best of our knowledge, this thesis is one of the first works in providing a formal definition of views. In previous works, only the view concept is elaborated and not defined. In this chapter, we formally define the view mechanism (and its properties) that is adaptable to not only XML, but also to other data models. In addition, here, we formally define one of the unique characteristics of the view mechanism; the *conceptual view operators*.

Chapter 6 and 7 discuss view specification, modelling and transformation issues associated with the XML views, demonstrated by using two OO modelling languages. Both chapters provide an illustrative walkthrough of the unique characteristics of XML views using a real-world industrial case study.

Chapter 6 looks at modelling and transformation methodologies of our XML views, using a semantic network based Object-Oriented modelling notation (Feng, Chang & Dillon 2002) called eXtensible Semantic (XSemantic) nets. Here, the XML view design methodology is systemically elaborated with detailed discussion on XSemantic nets and its various modelling notations. In addition, this chapter provides special remarks on the advantages and benefits of using the XSemantic nets for modelling XML domains. Also included is a discussion of modelling and transforming conceptual operators and dynamic view properties (such as *generic methods*) to XML query expressions.

Chapter 7 looks at another modelling and transformation methodology of our XML views, using an industry standard OO modelling language: the OMG's Unified Modelling Language (UML) (OMG-UML™ 2003). Here, special attention is given to Object Constraint Language (OCL) (OMG-OCL 2003) to complement UML modelling constructs, to explicitly specify view constraints, as UML lacks the ability to visually describe semi-structured data semantics and constraints in detail (Feng, Chang & Dillon 2003).

The following three chapters form the third part of this thesis. They look at data modelling issues associated with three different domains, where our XML views are utilized to provide *architectural frameworks* to solve issues in; (a) XML data warehousing, (b) ontology bases and views and (c) website (and web portal) development. Each chapter provides a comprehensive background information related to research in question (i.e. (a) to (c) above) and provide detailed discussion on how the XML views are utilized in developing the frameworks in the related domains, with concepts, definitions and formal semantics.

Chapter 8 looks at designing a view-driven XML warehouse model that is intuitive and comprehensive in modelling dimensional data. The XML warehouse presented here is a top-down conceptual model with three layers abstraction.

Chapter 9 discusses the emerging view model requirement in the Semantic Web (SW) (W3C-SW 2005) paradigm. First, a brief discussion on SW and ontology base is presented together with required characteristics of a view model for SW. Then, analogous to the views in XML, a notion of view concept and semantics are given in the context of ontologies. This chapter also includes a discussion on transforming and deploying the proposed ontology views in the Materialized Ontology View Extractor (MOVE) (Wouters 2006; Wouters et al. 2004a) system.

Chapter 10 presents an intuitive approach to modelling and deploying websites and web ports using XML views. Titled extensible Web (or *x*Web) and extensible portal (or *x*Portal), they are *architectural frameworks* with accompanying design methodologies that is capable of modelling websites (and web portal) using the top-down OO approach.

Chapter 11 provides a summary of the results achieved and an insight into the future work and research directions.

References

Aberer, K, Catarci, T, Cudré-Mauroux, P, Dillon, TS, Grimm, S, Hacid, M-S, Illarramendi, A, Jarrar, M, Kashyap, V, Mecella, M, Mena, E, Neuhold, EJ, Ouksel, AM, Risse, T, Scannapieco, M, Saltor, F, De Santis, L, Spaccapietra, S, Staab, S, Studer, R & De Troyer, O 2004, 'Emergent Semantics Systems', *First International IFIP Conference on Semantics of a Networked World (ICSNW 2004) - Revised Selected Papers*, vol.

3226, ed. CAG Mokrane Bouzeghoub, Vipul Kashyap, Stefano Spaccapietra, Springer, Paris, France, pp. 14-43.

Abiteboul, S 1999, 'On Views and XML', *Proceedings of the eighteenth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems (PODS '99)*, ACM Press New York, NY, USA, Philadelphia, Pennsylvania, USA, pp. 1-9.

Abiteboul, S, Amann, B, Cluet, S, Eyal, A, Mignet, L & Milo, T 1999, 'Active Views for Electronic Commerce', *Proceedings of the 25th International Conference on VLDB*, Edinburgh, Scotland, pp. 138-49.

Abiteboul, S & Bonner, A 1991, 'Objects and Views', *ACM SIGMOD Record, Proceedings of the International Conference on Management of Data (ACM SIGMOD '91)*, vol. 20 (2), ACM Press New York, NY, USA.

Abiteboul, S, Buneman, P & Suciu, D 1999, *Data on the Web : from relations to semistructured data and XML*, Morgan Kaufmann, London, UK.

Abiteboul, S, Goldman, R, McHugh, J, Vassalos, V & Zhuge, Y 1997, 'Views for Semistructured Data', *Workshop on Management of Semistructured Data*, USA.

Abiteboul, S, Quass, J, McHugh, J & Widom, J 1997, 'Lore: A Database Management System for Semistructured Data', *SIGMOD Record*, vol. 26(3), ACM, pp. 54-66.

Abiteboul, S, Quass, J, McHugh, J, Widom, J & Wiener, J 1997, 'The Lorel Query Language for Semistructured Data', *International Journal on Digital Libraries*, vol. 1, no. 1, pp. 68-88, April 1997.

Aguilera, V, Cluet, S, Milo, T, Veltri, P & Vodislav, D 2002, 'Views in a Large-Scale XML Repository', *The VLDB Journal —The International Journal on Very Large Data Bases*, vol. 11(3), no. 3, pp. 238-55

Bertino, E 1992, 'A View Mechanism for Object-Oriented Databases', *3rd International Conference on Extending Database Technology (EDBT '92)*, vol. 580, eds A Pirotte, C Delobel & G Gottlob, Springer, Vienna, Austria, pp. 136-51.

Braga, D, Campi, A & Ceri, S 2005, 'XQBE (XQuery By Example): A visual interface to the standard XML query language', *ACM Transactions on Database Systems (TODS)*, vol. 30(2), no. 2, pp. 398-443

Ceri, S, Comai, S, Fraternali, P, Paraboschi, S, Tanca, L & Damiani, E 1999, 'XML-GL: A Graphical Language for Querying and Restructuring XML Documents', *SEBD 1999*, Como, Italy, pp. 151-65.

Chan, H, Lee, R, Dillon, TS & Chang, E 2001, *E-Commerce Principles and Practice*, John Wiley & Sons, UK.

Chan, S, Dillon, TS & Siu, A 2002, 'Applying a mediator architecture employing XML to retailing Inventory Control', *The Journal of Systems and Software*, vol. 60, pp. 239-48

Chang, E 1996, 'Object Oriented User Interface Design and Usability Evaluation', Doctor of Philosophy (Ph.D) thesis, La Trobe University, Melbourne, Australia.

Chang, E & Dillon, TS 1994, 'Integration of User Interfaces with Application Software and Databases Through the Use of Perspectives', *1st International Conference on Object-Role Modeling (ORM '94)*, Australia, pp. 4-6.

Chen, YB, Ling, TW & Lee, ML 2002, 'Designing Valid XML Views', *Proceedings of the 21st International Conference on Conceptual Modeling (ER '02)*, vol. 2503, eds S Spaccapietra, ST March & Y Kambayashi, Springer-Verlag London, UK, Tampere, Finland, pp. 463 - 78.

Cluet, S, Veltri, P & Vodislav, D 2001, 'Views in a Large Scale XML Repository', *Proceedings of the 27th VLDB Conference (VLDB '01)*, Roma, Italy.

Codd, EF 1983, 'A relational model of data for large shared data banks', *Communications of the ACM*, vol. 26(1), no. 1, pp. 64-9

Codd, EF 1990, *The Relational Model for Database Management: Version 2*, Addison Wesley Publishing Company.

Conrad, R, Scheffner, D & Freytag, JC 2000, 'XML conceptual modeling using UML', *19th International Conference on Conceptual Modeling (ER '00)*, vol. 1920, eds AHF Laender, SW Liddle & VC Storey, Springer-Verlag London, UK, USA, pp. 558-71.

Date, CJ 2003, *An introduction to database systems*, 8th edn, Pearson/Addison Wesley, New York.

Date, CJ & Darwen, H 1998, *Foundation for object/relational databases : the third manifesto : a detailed study of the impact of objects and type theory on the relational model of data including a comprehensive proposal for type inheritance*, Addison-Wesley, Reading, Mass.

Dillon, TS & Tan, PL 1993, *Object-Oriented Conceptual Modeling*, Prentice Hall, Australia.

Do, HH & Rahm, E 2004, 'Flexible integration of molecular-biological annotation data: The genmapper approach', *Proceedings of the 9th International Conference on Extending Database Technology (EDBT '04)*, Heraklion, Crete, Greece.

Elmasri, R & Navathe, S 2004, *Fundamentals of database systems*, 4th edn, Pearson/Addison Wesley, New York.

Feng, L, Chang, E & Dillon, TS 2002, 'A Semantic Network-based Design Methodology for XML Documents', *ACM Transactions on Information Systems (TOIS)*, vol. 20, No 4, no. 4, pp. 390 - 421

Feng, L, Chang, E & Dillon, TS 2003, 'Schemata Transformation of Object-Oriented Conceptual Models to XML', *International Journal of Computer Systems Science & Engineering*, vol. 18, No. 1, no. 1, pp. 45-60

Feng, L & Dillon, TS 2005, 'An XML-Enabled Data Mining Query Language XML-DMQL', *International Journal of Business Intelligence and Data Mining*,

Feng, L, Dillon, TS, Weigand, H & Chang, E 2003, 'An XML-Enabled Association Rule Framework', *14th International Conference on Database and Expert Systems Applications (DEXA '03) 2003*, vol. 2736, ed. WR Vladimír Marík, Olga Stepánková, Springer-Verlag Heidelberg, Prague, Czech Republic, pp. 88 - 97.

Florescu, D 2005, 'Managing Semi-Structured Data', *ACM Queue*, October, pp. 18-24.

Gopalkrishnan, V, Li, Q & Karlapalem, K 1999, 'Star/Snow-flake Schema Driven Object-Relational Data Warehouse Design and Query Processing Strategies', *1st First International Conference on Data Warehousing and Knowledge Discovery (DaWaK '99)*, Springer, Florence Italy.

Graham, I, Wills, AC & O'Callaghan, AJ 2001, *Object-oriented methods : principles & practice*, 3rd edn, Addison-Wesley, Harlow.

Gupta, A, Mumick, IS & (eds) 1999, *Materialized views: techniques, implementations, and applications*, eds A Gupta & IS Mumick, MIT Press.

Hadzic, M & Chang, E 2004, 'Role of the Ontologies in the Context of Grid Computing and Application for the Human Disease Studies', *Semantics for Grid Databases, First International IFIP Conference on Semantics of a Networked World (ICSNW '04)*, vol. LNCS 3226, Springer Verlag, Paris, France, pp. 316-8.

Humphris, M, Hawkins, MW & Dy, MC 1999, *Data Warehousing: Architecture & Implementation*, Prentice Hall PTR, USA.

IEEE 2004, *IEEE Xplore®: http://ieeexplore.ieee.org*, IEEE.

Inmon, WH, Imhoff, C & Battas, G 1996, *Building the operational data store*, John Wiley & Sons, New York, USA.

ITEC 2002, *iPower Logistics (http://www.logistics.cbs.curtin.edu.au/)*, http://www.logistics.cbs.curtin.edu.au/.

Jeong, E & Hsu, C-N 2001, 'Introduction of Integrated View for XML Data with Heterogenous DTDs', *Proceedings of the 10th International Conference on Information and Knowledge Management (CIKM '01)*, ACM, Atlanta, Georgia, pp. 151-8.

Kim, W 1990, 'Research Directions in Object-Oriented Database Systems', *Proceedings of the ninth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, ACM Press New York, NY, USA, Nashville, Tennessee, USA, pp. 1-15.

Kim, W & Kelly, W 1995, 'Chapter 6: On View Support in Object-Oriented Database Systems', in *Modern Database Systems*, Addison-Wesley Publishing Company, pp. 108-29.

Kimball, R & Ross, M 2002, *The data warehouse toolkit : the complete guide to dimensional modeling*, 2nd edn, Wiley, New York.

Lucie-Xyleme 2001, 'Xyleme: A Dynamic Warehouse for XML Data of the Web', *International Database Engineering & Applications Symposium (IDEAS '01)*, eds ME Adiba, C Collet & BC Desai, IEEE Computer Society 2001, Grenoble, France, pp. 3-7.

Luk, RWP, Leong, HV, Dillon, TS, Chan, ATS, Croft, BW & Allan, J 2002, 'A survey in indexing and searching XML documents', *Journal of the American Society for Information Science and Technology*, vol. 53(6), no. 6, pp. 415-37

Mohania, MK, Karlapalem, K & Kambayashi, Y 1999, 'Data Warehouse Design and Maintenance through View Normalization', *10th International Conference on Database and Expert Systems Applications (DEXA '99)*, vol. 1677, Springer, Florence, Italy, pp. 747-50.

Nunamaker, JF, Chen, JM & Purdin, TDM 1991, 'System Development in Information Systems Research', *Journal of Management of Informatin Systems*, vol. 7(3), no. 3, pp. 89-106

OMG-MDA 2003, *The Architecture of Choice for a Changing World®, MDA Guide Version 1.0.1 (http://www.omg.org/mda/)*, OMG, 2005.

OMG-OCL 2003, *UML 2.0 OCL Final Adopted specification (http://www.omg.org/cgi-bin/doc?ptc/2003-10-14)*, OMG, 2005.

OMG-UML™ 2003, *UML 2.0 Final Adopted Specification (http://www.uml.org/#UML2.0)*.

Ponniah, P 2001, *Data Warehousing Fundamentals: A Comprehensive Guide for IT professionals*, John Wiley & Sons Inc., NY.

Rafanelli, M & (ed) (eds) 2003, *Multidimensional Databases: Problems and Solutions*, Idea Group Inc.

Springer 2005, *SpringerLink: http://www.springerlink.com*, Springer.

Tan, H, Dillon, TS, Hadzic, F & Feng, L 2005, 'MB3-Miner: mining eMBedded subTREEs using Tree Model Guided candidate generation', *The 1st International IEEE Workshop on Mining Complex Data (MCD '05), In conjunction with ICDM '05)*, IEEE Computer Society, Houston, Texas, USA.

Volz, R, Oberle, D & Studer, R 2003, 'Views for light-weight Web ontologies', *Proceedings of the ACM Symposium on Applied Computing (SAC '03)*, ACM Press New York, NY, USA, USA, pp. 1168 - 73.

W3C-DTD 2001, *XML Schema (http://www.w3.org/*, W3C, 2005.

W3C-HTML 1997, *HyperText Markup Language (HTML), Rel 4.01 (http://www.w3.org/MarkUp/)*, The World Wide Web Consortium (W3C).

W3C-OWL 2004, *OWL: Web Ontology Language 1.0 reference (http://www.w3.org/2004/OWL/)*, W3C, viewed 2005.

W3C-RDF 2004, *Resource Description Framework (RDF), (http://www.w3.org/RDF/)*, The World Wide Web Consortium (W3C), viewed March 2000.

W3C-RDQL 2004, *RDQL - A Query Language for RDF, (http://www.w3.org/Submission/2004/SUBM-RDQL-20040109/)*, W3C.

W3C-SW 2005, *The Semantic Web (http://www.w3.org/2001/sw/)*, W3C.

W3C-WS 2002, *Web Services Activity, (http://www.w3.org/2002/ws/)*, W3C.

W3C-WWW 1997, *World Wide Web (WWW), http://www.w3.org/*, The World Wide Web Consortium (W3C), http://www.w3.org/.

W3C-XML 2004, *Extensible Markup Language (XML) 1.0, (http://www.w3.org/XML/)*, The World Wide Web Consortium (W3C), http://www.w3.org/XML/.

W3C-XQuery 2004, *XQuery 1.0: An XML Query Language (http://www.w3.org/TR/xquery)*, The World Wide Web Consortium (W3C), viewed November 2003.

W3C-XSD 2001, *XML Schema (http://www.w3.org/XML/Schema)*, W3C, 2004.

W3C-XSL 2003, *Extensible Stylesheet Language (XSL) (http://www.w3.org/Style/XSL)*.

Wollersheim, D & Rahayu, JW 2005, 'Ontology Based Query Expansion Framework for Use in Medical Information Systems', *International Journal of Web Information Systems*, vol. 1, no. 2, pp. 101-15

Wouters, C 2006, 'A Formalization and Application of Ontology Extraction', Doctor of Philosophy (Ph.D) thesis, La Trobe University, Melbourne, Australia, Melbourne.

Wouters, C, Dillon, TS, Rahayu, JW, Chang, E & Meersman, R 2004a, 'Ontologies on the MOVE', *9th International Conference on Database Systems for Advanced Applications (DASFAA '04)*, vol. 2973/2004, eds Y Lee, J Li, K-Y Whang & D Lee, Springer-Verlag GmbH, Jeju Island, Korea, pp. 812-23.

Wouters, C, Dillon, TS, Rahayu, JW, Chang, E & Meersman, R 2004b, 'A Practical Approach to the Derivation of a Materialized Ontology View', in D Taniar & W Rahayu (eds), *Web Information Systems*, Idea Group Publishing, USA.

XMark 2003, *XMark —An XML Benchmark Project, (http://monetdb.cwi.nl/xml/)*.

Xyleme 2001, *Xyleme Project (http://www.xyleme.com/)*, http://www.xyleme.com/.

Zhuge, Y & Garcia-Molina, H 1998, 'Graph structured Views and Incremental Maintenance', *Proceeding of the 14th IEEE Conference on Data Engineering (ICDE '98)*, IEEE, USA.

Chapter 2

Evaluation of the Existing Literature on Views

2.1 Introduction

This chapter discusses the general background and research in the area of view formalism, as well as existing work and outstanding issues in the areas related to modelling, designing, specifying and defining of views in general. Though the notion of view concept is relatively old, dating back to the early relational data modes, there still remain many unexplored aspects such as high-level view design, semantics and applications of views outside the scope of data manipulating and database programming. The main issues and background information that are presented in this chapter will lead to the formal definition of the problem addressed in this thesis in Chapter 3.

The first part of this chapter (section 2.1) provides general information on views, view characteristics, evolution and its applications. The second part, sections 2.2 – 2.5, deals mainly with existing view formalisms to this date, related research and their shortcomings using various data models as a guide in view evolution. The third part (section 2.6) provides a brief outline of emerging new requirements and expected characteristics for view mechanisms with changing data paradigm (from structured data models to semi-structured and web data) as a case in point. The fourth part provides a discussion of the unexplored domain of view characteristics that are not sufficiently addressed in the previous sections. This includes a discussion of alternative view concepts (section 2.7.1), high-level view specification and representation (section 2.7.2), view constraint specification (section 2.7.3) and a summary of view-related research (section 2.7.4). Section 2.8 concludes this chapter.

2.2 The Notion of Views

The notion of view originated and was used in the early relational databases as a *subschema*; that is, a portion of the conceptual database or an abstraction of part of the conceptual database (Ullman 1989). It was originally intended for the purpose of database security and was later extended to support data abstraction to provide different (abstract) perspectives in a database environment to the end-users and end-user applications over the stored data without creating new data stores. Due to its persistent nature, views are also used as shorthand for queries, that is, user query expression is stored as part of the database (with a given name) and retrieved on request, similar to a data store. Thus, this created an informal definition of views, namely, the *named query expression*. However, only the query expression is stored and not the query result (i.e. query data set or instance) and only materializes (query execution) when the query is explicitly called (i.e. executed) or other queries are issued against it. To date, in any given data model or paradigm, this is the only notion of views, that is, it refers to a named query expression.

Views may be considered as an extended 2-E transformation applied on the stored data, that is, data Extraction and data Elaboration. In some paradigms (e.g. Semantic Web (W3C-SW 2005)), views may be considered as a 3-E transformation (Wouters 2006), that is, extraction, elaboration and extension applied on the stored data. However, most researchers agree on extracting and elaboration, as current views formalisms provide data extraction that is customized to the end-user (or application) requirements with added constraints or restriction applied on the extracted data (elaboration). Thus, in this research, our discussions focus mostly on the 2-Es aspects of views, unless explicitly stated otherwise.

Over the years, due to development and advancements in database systems, data models, and paradigms, today we have many different data models to support different paradigms from relational to, semi-structured to Semantic Web. Thus, based on these developments, we can group the existing data models into five broad categories, namely:

(i) classical (or relational) data models

(ii) Object-Oriented (OO) data models

(iii) semi-structured data models

(iv) Semantic Web and ontological models

Therefore, from the data models stated in (i) – (iv) above, we can also group the existing view formalisms into such broad categories, namely:

(i) Views in the relational model

(ii) Views in for OO data models

(iii) Views for semi-structured data models (mainly XML)

(iv) Views for Semantic Web paradigm

In the following sections, we will present a detailed discussion of the above view models focusing mainly on:

(i) View characteristics

(ii) View specification and definition

(iii) View design

(iv) View constraint specification

But we do not provide discussion of view storage models, performance issues and tuning as it is outside the scope of this research. An extensive set of literature can be found in both academic and industry forums in relation to these issues.

2.3 Views for Relational Model

Since its introduction, the relational view formalism (Codd 1970; Date 2003; Elmasri & Navathe 2004; Ullman 1989) has been studied extensively in the context of:

(i) Database views for data elaboration

(ii) User access control

(iii) Dimensional data modeling in data warehouse environments

(iv) Performance

(v) View updates

(vi) View updates data warehouse environments

(vii) Query refinements and tuning

Since the early relational models, though the importance of views in the relational model was realized in the context of user access control, it was re-visited and further research directions were taken later in the context of data warehouses. The notion of views was useful and essential for defining and representing aggregate queries, OLAP queries and complex query design of data mining warehouses.

A view in the relational database provided data "abstraction" and data "partitioning". It is specified and defined using language specific data manipulation statement using the relational query language ANSI SQL. It is usually a Select-Project-Join query that provides vertical and horizontal partitioning of the stored relations as required.

There exist two versions of the interpretation of the relational views. The first (and the earliest) interpretation of the relation view is that, as stated earlier, it is a portion or an abstraction of the conceptual database (Ullman 1989). Here, the usage of the notion is different from the notion considered today (and in this research). Here, as in the old relational model, a conceptual database is defined as an abstraction of the real-world as perceived by a database user (Ullman 1989). Here, the word "database" should be notated as, today, the concept of a database is not considered as part of the conceptual model but as part of the deployment or implementation model concept (Dillon & Tan 1993).

A modern and widely accepted interpretation (and implemented) of the relational view is that, relational databases views (Codd 1990) are part of the external schema of the ANSI three-schema (Tsichritzis & Klug 1978) (i.e. conceptual, storage and external) architecture[1] (Date 2003; Kim & Kelly 1995), where persistent view definitions are instantiated when a query is issued against the view. Conversely, new view definitions can be created and/or deleted without affecting the consistency, structure and storage space of the overall database schemas and data.

Another important aspect in regards to relational view is the modelling and design views. For a long time, conceptual modelling of relational data is strongly coupled with the Entity-Relationship Diagram (ERD) (Chen 1976; Date 2003; Dillon & Tan 1993; Elmasri & Navathe 2004) and Data Flow Diagrams (DFD), where data is always perceived as entities, attributes and relationships, which closely resemble relational tables. In an ERD, the notion of entity closely resembles a table in the relational database. Thus, the modelling is restricted by this notion and other modelling aspects that are not properties of a "table", such as constraints (except cardinality constraints), behaviour, views, etc are not represented at this level. Also, the logical model consists of relational schemas of the relations that are to be deployed in the database. Thus, we say that the relational views are not first-class citizens of the conceptual model.

Here, the conceptual models (i.e. ERD and DFD) are usually non-hierarchical and represent a set based tuples. The relationships are maintain using named properties such as keys (primary key and foreign key and composite keys), which are explicitly stated in the ERD. However, the notion of views has no conceptual or logical level representation and is never considered as part of the conceptual or logical model concept. It has always been a relational query language concept (i.e. SQL), as part of the data manipulation language (DML) construct.

In theory, a relational view is considered to be a collection of *virtual tuples* that form a *virtual relation* (or table) (Codd 1970, 1983, 1990; Date 2003; Elmasri & Navathe 2004). Also, the view definitions are expressed only in terms of (Codd 1990),

[1] The 3-Schema Architecture or ANSI/SPARC Architecture.

base or stored relations and other virtual relations. Usually, relational views are defined using the SQL '92 (Date 2003; Elmasri & Navathe 2004) compatible relational query languages and the view construction, materialization and updates are managed by the Database Management System (DBMS). View constraints (i.e. domain constraints, tuple type restriction, etc) are usually specified by using the SQL language syntax and restricted by the vendor specific functionalities available to the DBMS. In some instances, DBMS allow external program models to define specify view (and database) such constraints, such as in Oracle[2] (PL/SQL, SQL/C++, etc.), IBM DB2[3] systems.

In addition, in the early days, since the early conventional relational data model was restricted to only base data types, the presumed usage for views was to provide data protection against unauthorized user access. Later, when relational data models were extended to support complex data types and functions, the concept of views was extended to support selective updates (i.e. using view updates), data partitioning (vertical and horizontal), data integration, complex and aggregate/summary queries (such as data warehouse queries).

Also, a relational model is not organized in a specific hierarchy. In a given relational schema, all relations are defined at the same semantic level. They are equivalent to a flat set, grouped into relevant relations. Thus, in the case of view specification and definition, one can define relational views as:

(a) using base or stored relations

(b) using existing virtual relations (i.e. views)

(c) using both stored and existing virtual relations

(d) using new derived relations (that is, using new derived column definitions)

(e) using any combination of (a) to (d) above.

[2] Oracle DBMS, http://www.oracle.com/technology/products/database

[3] IBM DB2, http://www-306.ibm.com/software/data/db2/

Thus, in relational views, there is no concept of relation hierarchy and usually one can mix stored relations and virtual relation properties in defining or specifying new virtual relations. However, the abovementioned (a) to (b) are usually DBMS specific (not SQL) and restricted by the underlying vendor specific relational storage model.

It should be noted here that, nested relations are of type forced hierarchy. That is, the relation hierarchies are maintained externally to the relation (or table) semantics to emulate relation hierarchy, which is not considered as (and not modelled) as hierarchical relationships, used in the context of OO models.

Also, in developing views for XML, though the relational model is proven for its matured standards, query performance and simplicity, it is not an ideal platform for XML views as:

(i) The relational model is flat data structure and requires nested types to represent native XML document structure, thus increase the complexity and impedes the performance.

(ii) Mapping conceptual semantics captured in the domain XML are difficult to map to relations without loss of semantics. Thus, the XML view constructed may not have certain semantics from the original domain, thus presents an incomplete view definition.

(iii) The relational model is based on the concept of keys (primary / foreign.

(iv) The mapping of an XML document hierarch to relational data may result in un-normalized and/or meaningless tables, thereby reducing the integrity of the database.

(v) It is difficult to maintain an XML document hierarchy within a relational model with frequent changes.

(vi) Also, though the SQL standards are extended to support OO like features (SQL 3) and XML (SQL 2003 / SQL X (ANSI & ISO 2003)), to define views with support for conceptual and schemata semantics may increase the complexity of view definition with further language extensions required.

In the next section we look at some of the extended view formalisms that are based on the relational view concept.

2.4 Views for Object-Oriented Paradigm

During the OO revolution and data model proposal, the relational view definitions were extended to OO data models first by Won Kim et al. (Kim 1990; Kim & Kelly 1995), Abiteboul et al. (Abiteboul & Bonner 1991), and Chang (Elizabeth Chang 1996). Here, the views were defined in a synonymous manner to the relational view formalism by extending the relational definition (Kim 1990) when needed to support the 2-Es. They included the notion of *virtual class*. However, unlike the case of relational views, the issue of class and virtual class is the fundamental issue in OO views, which to this day remains a point of argument and is addressed in detail later in this section.

In the OO paradigm, a class has both attributes (may be nested or set valued) and methods. A class can also form complex hierarchies (inheritance, composition, etc) with other classes. Owing to these complexities, a definition of views for OO paradigm is not straightforward. As described, in one of the early discussions on OO views, Won Kim et al. (Kim 1990) argues that an OO data model should be considered as one kind of extended relational view formalism. Naturally, this statement later reflected on most discussions concerned with OO views in (Abiteboul & Bonner 1991; Elizabeth Chang 1996; Kim 1990; Kim & Kelly 1995).

In general, the concept of *virtual class* (as opposed to base, stored or domain class) is considered as the OO equivalent of the relational view formalism (virtual relation). Both relational and OO view concepts make two implicit assumptions: that the underlying data is structured, and there exists a fixed data model and a data access/query language. This, we argue, is not enough to provide a real-world scenario and/or abstraction to complex domain. We argue that providing view formalism at the conceptual level will improve the resulting view implementation, similar to a conceptual model of a software system.

In regards to abstraction (or modelling) of views, similar to relational views, only the work of Chang et al. (Elizabeth Chang & Dillon 1994) allows some form of abstraction at a higher level, a view definition in the form of *abstract views* (Elizabeth Chang 1996; Elizabeth Chang & Dillon 1994). All other view definitions are defined at the data manipulation language level, analogous to the relational views. That is, OO views are not *first-class citizens* of the conceptual model. In one of the recent works (Alhajj, Polat & Yilmaz 2005), the authors consider treating OO views as first-class citizens in the OO database schema. This is different from our discussion, where we refer to views as *first-class* citizens in the OO conceptual model.

In regards to virtual class hierarchy and instances, Abiteboul and Bonner (Abiteboul & Bonner 1991) argue that virtual classes and stored classes are interchangeable in a class hierarchy. They also stated that, virtual classes are populated by creating new objects (imaginary objects) from existing objects and other classes (virtual objects).

However, Won Kim et al. and Chang et al. argue that, in contrast to relational models, class hierarchy and view (virtual) class hierarchy should be kept separate (Elizabeth Chang 1996; Kim & Kelly 1995). It is inappropriate to include the view virtual class in the domain inheritance hierarchies because:

(i) A virtual class (view) can be derived from an existing class by having fewer attributes and more methods. It would be inappropriate to treat it as a subclass unless one allows for the notion of selective inheritance of attributes.

(ii) Two views could be derived from the same subclass with different groups of instances. However, the instances from one view definition could be overlapping with the other and non-disjoint.

(iii) View definitions, while useful for examining data, might give rise to classes that may not be semantically meaningful to users (Bertino 1992).

(iv) Effects of schema changes on classes are automatically propagated to all subclasses. If a view is considered as a subclass, this could create problems (Kim & Kelly 1995) in requiring the changes to be propagated to the view as it might be appropriate or inappropriate.

(v) An inappropriate placement of the view in the inheritance hierarchy could lead to the violation of the semantics because of the extent of overlapping with an existing class (Elizabeth Chang 1996; Kim & Kelly 1995).

Therefore, from the arguments provides above, in this research, whenever we refer to a virtual class hierarchy, unless explicitly stated, we consider keeping the domain class hierarchy and the virtual class hierarchy separate.

The view notion in OO (virtual class) described above formed the basis for many other works, such as MultiView system (Kuno & Rundensteiner 1993), views in OO databases (Alhajj, Polat & Yilmaz 2005), Active Views (Abiteboul et al. 1999) and views for semi-structured data (Abiteboul, Goldman et al. 1997; Aguilera et al. 2002; Cluet, Veltri & Vodislav 2001).

The MultiView system is a framework for the specification, creation and management of updatable OO views in the OO database systems (Kuno & Rundensteiner 1993). The implementation of the MultiView system extends the commercially available GemStone OODB model by employing a three-layered wrapper, with meta-objects, schema-objects and data objects. It considers the following view related tasks as part of the view specification:

(i) Class derivation

(ii) Global schema integration

(iii) View class selection

(iv) View hierarchy generation

The system supports definitions of views through user-queries and automatically organizes both the domain and view class hierarchy into one global schema, a noted drawback as described above. The MultiView system, though, clearly distinguish views and the associated instances from the base classes and instances; it organize both base classes and view classes into one global hierarchy. This, as described above, causes many problems. Also, it uses one global schema integrating both views and base classes.

Another problem foreseen in the MultiView system is that it implements a class as a type. Though classes and types are unteachable at the conceptual level (Dillon & Tan 1993), they may cause concerns at the implementation levels, especially at run-time, as a type base system usually does not provide run-time feedback (as opposed to type checking at design time). Also, views in the MultiView system are available only at the database schema and query level and no mechanism to visualise the views at the conceptual level is possible.

In one of the very recent works in (Alhajj, Polat & Yilmaz 2005), the authors present an automated process to place virtual classes in an appropriate position, independent of the user knowledge of the available hierarchies in an OO database system. Here, the authors argue that the OO views should be treated as first-class citizen of the OO databases. However, as based on similar arguments present above, though support for dynamic environment is provide by this work, the maintenance of such a complicated, system maintained mixed class hierarchies with updatable option is a not feasible solution in the wrong run as many meaningless classes (Bertino 1992) and semantics may be accumulated, especially in a dynamic environment.

One of the notable works that is based utilizes OO view concepts is the Active View system (Abiteboul et al. 1999) for e-Commerce. The Active View system uses database like features including (e.g. views) and active rules or triggers to support e-Commerce activities, such as access control, activity logging, etc. It uses OO views concepts and uses XML (W3C-XML 2004) technologies to specify and describe view concepts. Since it is XML based, the discussion of this is provided in the next section. The characteristics of the Active View system include:

(i) the view and class hierarchies are allowed to mix, a drawback as described above, especially e-Commerce being a dynamic environment;

(ii) view speciation is done using declarative languages;

(iii) deployed using Java language and XML as the data model, thus expressive and flexible;

(iv) views are constructed based on active rules; and

(v) the specification and definition of views are is only available at the implementation level.

In this section, based on the existing work, we described the notion of views in the OO paradigm. We also provided detailed a discussion of the differences, concerns and the current research directions that exist for the OO views. However, similar to the relational model, using the OO model for XML views is not an ideal solution as, though expressive, by nature, OO models are complex to deal with. Therefore, using it to store XML data (and views) may add further complexity. In addition, there is no defined standard for querying XML in the OO environments. Another drawback of using OODB models for XML is the issue of query performance, as good OO queries are complex to write. In the next section we will look at views formalisms that are proposed for the semi-structured data.

2.5 Views for Semi-Structured Data

Since the emergence of semi-structured data models, namely XML (W3C-XML 2004), the need for semi-structured data models to be independent of the fixed data structure and description go against the fundamental properties of the classical *structured* data models. However, since its introduction, much work has been directed towards providing traditional database-like features such as data access, query language, views, etc. to XML in an ongoing process.

XML documents which are tag-based and self-describing data documents represent a hierarchical tree structure. At the conceptual level, they can be visualised as hierarchical trees or graphs. An XML document is usually associated with a Document Type Definition (DTD) (W3C-DTD 2001) or XML Schema (W3C-XSD 2001) which is used to define and constrain the syntax and structure of a document. Though XML is usually considered as semi-structured data, there exist some differences between the two.

Many researchers have attempted to resolve view related issues in the semi-structured paradigm by using graph based (Abiteboul, Buneman & Suciu 1999; Goldman, McHugh & Widom 1999; Zhuge & Garcia-Molina 1998) and/or OO based data models (Abiteboul et al. 2002a; Abiteboul, Goldman et al. 1997; Aguilera et al. 2002). But, in all the above cases, as in the case of relational and OO, the actual view specification and definitions are available only at the lower levels of abstraction (implementation level) and not at the conceptual and/or logical level. This is one of the main issues of concern for views in the semi-structured (and XML) data models.

Semi-structured data (from XML to Semantic Web), unlike structured data do not follow a rigid, well-defined structure (as opposed to relational and OO data) and usually constrained and described by schemas. For example, the popularity of XML is due to its ability to support heterogeneity of data and structure for a given concept. That is, at the schema level, XML may define and constrain an XML document, while allowing XML document to have incomplete information. In such situations, defining and specifying views using the classical approach, that is, using a data manipulation or query language, may have allow for flexibility of dealing with incomplete or missing information. This is a complex task in comparison to writing simple view definitions.

Conversely, at a higher level of abstraction than the data language level, XML (and semi-structured) data can be easily visualized, defined, cataloged, queried and customized to suit one's need (Abiteboul, Buneman & Suciu 1999; Feng, Chang & Dillon 2002; Feng, Chang & Dillon 2003; Volz, Oberle & Studer 2003b; Wouters 2006). Thus, providing a view formalism for semi-structured data should consider specifying views at a higher level of abstraction than relational or OO views.

2.5.1 Declarative XML Views

One of the early discussions on XML views was by Serge Abiteboul (Abiteboul 1999) and later more formally by Sophie Cluet et al. (Cluet, Veltri & Vodislav 2001). They proposed a declarative notion of XML views. Abiteboul points out that:

(i) a view for XML, unlike relational or OO views, should do more than just providing different presentation of underlying data (Abiteboul 1999). This, he argues, arises mainly due to the nature (semi-structured) and the usage (primarily as common data model for heterogeneous data on the web) of XML;

(ii) XML views should use OO views as the foundation, should follow analogous to (his (Abiteboul & Bonner 1991)) OO view concepts, such as virtual values, virtual classes and imaginary classes;

(iii) an XML view specification should rely on a data model (like ODMG (Cattell et al. 2000) model) and a query language. Query language, he staes as the central issue for view definition, as query languages for XML are still at their initial stages of development or still evolving, and no view creation facilities are provided. W3C XPath (W3C-XPath 1999) and XQuery (W3C-XQuery 2004) are such examples, together with the SQL 2003 standard;

(iv) an XML view specification should be more than relational and OO views as XML is deployed over web. Thus, web specific characteristics, such as replication, change notification, etc. should be part of the view specification;

(v) an XML view should allow for dealing with incomplete information.

The above points in Abiteboul's work identify some of the core issues that are concerns for view formalism for XML. However, his proposition to couple XML to a structured data model (ODMG) and to query languages, in our opinion may restrict the capabilities of XML and may be considers as an extended OO model for XML than a native XML view formalism. Also, the problems that are associated with the mixed OO class and view hierarchy is still an issue here in these proposed XML views.

2.5.2 View Mechanism in Xyleme

In the works (Aguilera et al. 2002; Cluet, Veltri & Vodislav 2001), authors extend some of the concepts proposed in (Abiteboul 1999), and proposed a development of a view formalism for XML to support web scale XML data warehouse environment: the Xyleme (Lucie-Xyleme 2001; Xyleme 2001) project. In regards to the XML views, due to its scale, the authors propose a semi-automatic approach to view generation and they discuss in detail on how abstract XML paths/DTDs are mapped to concrete paths/DTDs. More formally an XML view definition in Xyleme is stated as (Cluet, Veltri & Vodislav 2001):

"....A view defined by a set of pairs <p,p'>, *called mappings, where* p *is a path in the abstract DTD and* p' *a path in some concrete DTD....."*

These concepts, which are implemented in the Xyleme project provides one of the most comprehensive mechanisms to construct an XML view to date. The Xyleme project uses an extension of ODMG Object Query Language (OQL) to implement such an XML view. But, in relation to conceptual modelling, these view concepts provide no support. The view model is derived from the instantiated XML documents (instant level) and is associated with DTD in comparison to flexible XML Schema. Also, the Xyleme view concept is mainly focused on web-based XML data. Characteristics of Xyleme views include (Aguilera et al. 2002; Cluet, Veltri & Vodislav 2001; Lucie-Xyleme 2001):

(i) due to the scale of Xyleme data set, XML views enables the user to query a single structure that summarizes a domain of interest (as opposed to multiple schemas);

(ii) Xyleme views operates on web scale clusters (as opposed to relational or OO data), XML data gathered by web crawlers that are automatically classified by external tools;

(iii) a view query in Xyleme is defined by a union of many queries over many clusters. Usually, here, the views are constructed automatically;

(iv) the query language for XML views in Xyleme is a variant (extension) of the OQL (Cattell et al. 2000) and also satisfy some of the XQuery features;

(v) the concepts related the view domain is considered as *concrete* and the concepts related to the XML view is considered *abstract* (e.g. DTD, values, etc.);

(vi) in regards to the properties of the XML view in Xyleme:

   a. View definition is a OQL like

   b. View is defined by a set if path pairs (or mappings) with a path in the abstract DTD and path in the concrete DTD

   c. View schema is provided by an abstract DTD

As stated earlier, XML views in Xyleme provide one of the most comprehensive implementation architectures for constructing web scale XML views. It is scalable and based on core OO data model principles. However, given the scale of the Xyleme project, it is desirable to have language-neutral, high-level, view specification and definition formalism. This will arguably provide a better transformation and mapping formalism to the existing domain specific XML document structures and language-specific view definitions. Also, a high-level view specification formalism will provide view visibility to end users rather than semi-auto generated view specification code. Another desirable feature for Xyeleme is the utilization of XML Schema as opposed to DTD as XML Schema is better suited for such web scale task (added deceptions and vocabularies, available data descriptors, support for multi-schema documents, etc.).

2.5.3 Views in the ORA-SS Model

Object-Relationship-Attribute Model for Semi-Structured Data (ORA-SS) is one of the early works that looked at view specification at a high-level abstraction (as opposed to query language) using visual constructs was proposed in (YB Chen, TW Ling & ML Lee 2002; YB Chen, TW Ling & M-L Lee 2002). This is also one of the first works that looks at using XML Schema and XQuery in specifying XML views. Here, XQuery is used as the view/query language over the views. The view definition includes an additional view declaration clause before XQuery expression.

The ORA-SS data model is a visual notation with schema, instance, and functional dependency and inheritance diagrams. It is comparable to the Extended-ER model with three basic concepts: object classes, relationship types and attributes. Here, an object class corresponds to an entity type an element in XML documents. A relationship type describes a relationship among object classes. Attributes are properties, and may belong to an object class or a relationship type. ORA-SS data model has four diagrams: the schema diagram, the instance diagram, the functional dependency diagram and the inheritance diagram. This is one of the first view models that support some of abstraction above the data language level.

The XML view in ORA-SS is done in the following steps:

(i) transformation of the stored XML documents into the ORA-SS model (diagram);

(ii) additional semantics that are not feasible to specify at the instance XML document level, but provided in the ORA-SS model are added, including: participation constraints of object classes and distinction between attributes of object classes and relationship types. Also, with end-user participation (manually) the following sub-tasks are performed:

   i. identification of key attributes of each object class;

   ii. identification attributes that belong to object classes;

   iii. identification of relationship types among object classes;

   iv. identification of attributes of relationship types.

(iii) The development of a set of rules to guide the design of valid XML views. These include, model specific four visual transformation operations for creating XML views, namely, selection, projection, join and swap operation.

(iv) Algorithm driven validation for the XML views constructed.

This is one of the early works that elaborates on the conceptual model like semantics to define XML views. However, the drawback this approach includes:

(i) ORA-SS model is constructed from instance level semantics; that is, it represents and XML document as opposed to representing conceptual or logical semantics that may have lost or not present in the XML document. Thus, this work may be analogous to Query-by-Example tools in relational or work such as (Augurusa et al. 2003; Braga & Campi 2003; Braga, Campi & Ceri 2005), where XML document semantics are used to create visual representation of the XML document semantics and used to construct XML views.

(ii) Though the view formalism provides semantic enrichment with user participation and explicit diagrammatic representation, it does not consider the addition of conceptual semantics or important constraints that are useful for the XML domain, such as ordered composition, exclusive disjunction, etc.

(iii) View definitions are possible only if the source data is accurate and complete. No provision is given in the ORA-SS model to deal with missing or incomplete XML node, element or attribute values.

(iv) Validation requires external algorithms.

2.5.4 Technology Specific View Formalisms

In this section, we look at some of the XML view proposals that are the by-product of another concept or technological solution. These include: Active XML (Abiteboul, Benjelloun & Milo 2004) view system and the MIX view (Ludaescher, Papakonstantinou & Velikhov 2000; Ludaescher et al. 1999) system. Though not comprehensive enough to be considered as an XML view concept, they do address some of the issues associated with XML views. The following sections look at this research in some detail.

2.5.4.1 Active XML Views

As described above in Active View systems, the Active XML (Abiteboul et al. 2002a, 2002b; Abiteboul, Benjelloun & Milo 2004; Benjelloun 2004) views are the result of using XML as the data model in the Active Views system. The Active Views system is built on top of Ardent Software XML repository based on the O2 OODBMS system (Benjelloun 2004). In the Active Views system, since view is presented as an object, it allows for both properties and methods. The Active Views system uses the OQL as a view definition language and the Lorel (Abiteboul, Quass, McHugh & Widom 1997; Abiteboul, Quass, McHugh, Widom et al. 1997) language as the query language over the views.

2.5.4.2 Views in the MIX System

The MIX View system (Ludaescher, Papakonstantinou & Velikhov 2000; Ludaescher et al. 1999) is a by-product of developing web scale mediator systems. It is based on supporting mediator architecture to provide the user with an integrated view of the underlying heterogeneous information/data sources. The MIX system employs XML as the data exchange and integration medium between mediator components and the XML DTD to provide structural descriptions of the data.

The Query language of MIX system is Xmas (Baru et al. 1998) (XML Matching and Structuring Language) which is claimed to be a high-level, declarative query language. The Xmas provides:

(i) object fusion and pattern matching on the input XML data; and

(ii) grouping and order constructs for generating new integrated XML objects.

The interface to MIX is provided by the graphical user interface BBQ (Blended Browsing and Querying) (Munroe & Papakonstantinou 2000) which support Query-by-Example operations. The output of a BBQ operation is a Xmas query. Though MIX system provides support for XML views, it is not a XML View by nature. It is a by-product to support data mediation for web-based information systems. Though powerful, the drawbacks include:

(i) no standalone framework to support XML views and non-standard language/(s) used to query/manipulate data;

(ii) it does not utilize XML Schema, a semantically rich replacement for XML DTD;

(iii) the resulting views are not systematically validated.

2.6 The Semantic Web Paradigm and Views

In this section, we present some concepts related to the new web paradigm, the Semantic Web (SW) and the view formalism that exists for this paradigm. The success of SW depends on successful formulation of ontology bases (Wouters 2006) and the interoperability of such vast vocabularies shared over the SW. However, to allow database-like features for SW (e.g. querying, views, etc.), it requires more than just the adaptation of the concepts (e.g. view, query, etc.), which is *not* in the case of OO, whereas in most cases, many of the concepts were "borrowed" from the relational model.

This is because ontology bases are metadata repositories described using metadata representation languages such as Resource Description Framework (RDF) (W3C-RDF 2004), OWL (W3C-OWL 2004) , RDF-S (W3C-RDFS 2004), etc., and queried for metadata and vocabularies and the relationships (as opposed to data instances or tuples). However, the underlying representation model for the SW languages are usually XML based (i.e. they are XML documents) and exhibits XML characteristics, such well-formness, validation, etc. As stated in (Volz, Oberle & Studer 2003b), SW enables access to distributed content using a standardized semi-structured data model at a higher level of abstraction than traditional data.

Another property of ontology engineering (Gâomez-Pâerez, Fernâandez-Lâopez & Corcho 2003; Spyns, Meersman & Mustafa 2002; Wouters 2006) that is different for the structured data is that, it is also possible to have instance level data and logical level schemata descriptors to inter-mix, which may present a type of mixed sorts, which is not the case in traditional structured data (and also usually in XML) data models. In addition, ontology bases (and related concepts such as ontology views) should provide support for *intensional* and *extensional* queries excuted over the base and view ontologies.

In this research, it is outside the scope of our work to present an in-depth analysis of the issues and concepts related to the SW. Our main focus in this research is the view formalism, namely for XML. However, some of the characteristics addressed in this thesis are also applicable and adaptable to the SW paradigm. As a case in point, we do provide such adaptation as part of our case study and applications in Chapter 9, where some of our research results are applied to SW where our extended research concepts were easily adapted and applied. Thus, it is imperative to understand some of the concepts and theories in regards to ontology view formalism (and related concepts) that exists for SW and later in this section we address some such research done in this area. For the interested reader, we direct them to non-view related, SW research directions that are of some interest to our research here (and in Chapter 9). They are:

(a) conceptual modelling of (base) ontologies using high-level OO languages (Gaševic, Djuric & Devedzic 2005; Dragan Gaševic et al. 2004a; Jarrar, Demey & Meersman 2003; OMG-ODM 2005; Wouters 2006);

(b) modelling, representing and transformation of base ontologies (Dragan Gaševic et al. 2004b; D. Gaševic et al. 2004);

(c) domain specific representation and applications of ontologies (Ceravolo et al. 2006; Ceravolo, Damiani & Viviani 2005; Elizabeth Chang, Dillon & Hussain 2006; Gâomez-Pâerez, Fernâandez-Lâopez & Corcho 2003; Hadzic & Chang 2004; Noy & Musen 2002; Wollersheim & Rahayu 2005; Wongthamtham, Chang & Dillon 2004).

The existing ontology view research ranges from low-level, implementation specific interpretations to high-level semantic aware view formalisms, such as in (Wouters 2006). These view researches are constrained by: (a) available ontology representation language or notation; (b) the lack of standardized storage model (and structure) for storing and querying ontology bases; and (c) the lack of standardized query languages. However, for ease of understanding, here, for ease of understanding, we broadly grouped the ontology view research into two categories based on the type of storage structure and/or data model adapted to store and represent the "stored" (or base) ontology base. They are:

(a) ontology views that defined over ontology bases in purpose built repositories for storing and querying ontologies. For example, the works that fall in this category includes, the MOVE view system (Wouters 2006; Wouters et al. 2004a), User Oriented Hybrid Ontology Development Environments (HyOntUse 2003), etc.;

(b) ontology views that are defined over ontology bases in OO (or relational) databases or extended OO/relational environments. Here, the works include, the CLOVE project (Uceda-Sosa, Chen & Claypool 2004), KAON project (KAON 2004), etc.

Now we will present some of this research in detail.

2.6.1 RDF Views

One of the first works that explored the notion of RDF views was in (Volz, Oberle & Studer 2003a, 2003b). It is a low-level, yet very practical approach combining RDF technologies and relational databases. It is a view formalism for RDF based documents with support for RDF schema (RDF-S) (W3C-RDFS 2004) using RDF schema supported query language called RQL (W3C-RDQL 2004). This is one of the early works that focused purely on the RDF/SW paradigm and has sufficient support for logical modelling of RDF views. A working prototype (Volz, Oberle & Studer 2003a) was developed and the extension (and ongoing research) of this work (and other related projects) can be found at (KAON 2004).

The original intended purpose was to enable describing and encoding of web resources using metadata. RDF is an object-attribute-value triple, where it implies object has an attribute with a value (Feng, Chang & Dillon 2003). It makes only intentional semantics and not data modelling semantics. Therefore, unlike generic view models, views for such RDF (both logical and concrete) have no tangible scope outside its domain. However, the work addresses some of the core issues related to view proposal for SW and provides a comprehensive implementation architecture (Volz, Oberle & Studer 2003a) to construct RDF views. Also, the proposed view hierarchy is analogous to the OO (is-a) hierarchy (i.e. the class and view hierarchies are not allow to mix) and a relational data is chosen as the storage structure for the RDF documents. An extension of this work, to extend the view language using a Description Logic Programs (DLP) and view updates is being considered (Volz, Oberle & Studer 2003a).

However, some of the drawback of this work includes:

(i) The view formalism is strongly coupled to RDF technologies and no provision is provided accommodate other, more elaborate metadata description languages such as OWL. Though the authors have proposed to extend the view language to a description logical based web ontology language (Volz, Oberle & Studer 2003a), it is time-consuming to port view definitions and specification from one language to another just because the language semantics, query language and encodings are different or changing. Thus, since ontologies are usually a high-level representation, it is more desirable to have view formalism in a generic form that is capable dealing with any current or upcoming SW language proposal.

(ii) Another sticking point is that the authors seemed to assume that the source RDF documents that are stored in a relational model are capable of handling all the possible conceptual semantics that is required to construct views (or sub-ontologies). The work does not address the issue of transfer (or loss) of conceptual semantics due to the mapping between conceptual model – to – RDF – to relational database.

(iii) As in the case of other view formalism (from relational to OO and SW) the view specification and definition is tied to query language and, given the notion of ontology bases and its level of abstraction, as authors themselves stated as the "explicit conceptual level semantics", proposing a view formalism with language and deployment specific semantics may cause problems in a shared community of heterogenous ontology (conceptual) environment.

(iv) Also, as stated by Wouters et al. in (Wouters 2006; Wouters et al. 2005; Wouters et al. 2004b), sub-ontology extraction should not be simplified as a query operations issued against a semi-structured data repository as in the case in this work.

(v) Another issue that is not clearly addressed here is the distinction between *generic* and *specific* ontologies (Elizabeth Chang, Dillon & Hussain 2006) and the related view definitions.

2.6.2 OWL Views

Another area that is currently under development is the view formalism for SW meta languages such as OWL. In some SW communities, OWL is considered to be a conceptual level modelling language for representing ontologies, while some others consider it to be a crossover language with rich conceptual semantics and RDF like schema structures (Wouters 2006; Wouters et al. 2004a). It is outside the scope of this paper to provide argument for or against OWL being a conceptual modelling language. Here, we highlight only one of view formalism that is *under development* for OWL, namely views for OWL in the "User Oriented Hybrid Ontology Development Environments (HyOntUse)" (HyOntUse 2003) project. The project proposed to use its core knowledge and expertise in developing domain specific, e-Science related ontologies in developing ontology bases that are simpler representations with the power of expressive ontology formalisms with ease of use.

Though still in development, one of the interesting points in this project is the proposed approach taken towards the notion of ontology views and the level of abstraction to consider. Here, the OWL (ontology) representations are considered analogously to "assembly languages", while the ontology views are analogous to the "high level programming languages". That is, the views are considered as an "Intermediate Representation" (HyOntUse 2003). This notion of ontology, in the context of abstraction levels, comes close to our notion of views considered in this research and that of ontology views considered in (Wouters 2006). However, the

HyOntUse is still a proposal and no specific information is available to evaluate the project further.

2.6.3 Ontology Views in the MOVE System

In a related area of research, the authors of the work Materialized Ontology View Extractor (MOVE) system (Wouters 2006; Wouters et al. 2002, 2005; Wouters et al. 2004a, 2004b) proposed an ontology view formalism for ontology extraction, with limited conceptual semantics and logical extensions, where materialized ontology views are formulated from stored ontologies.

Though the main work focuses on view-driven ontology extraction, the work provides formal definitions and concept representations using high-level OO modelling languages, such as OMG's UML (OMG-UML™ 2003b), semantic nets (Feng, Chang & Dillon 2002), etc. The work proposes a "restricted" ontology (and ontology view) definition, where high-level operators are proposed to define ontology view representations with limited conceptual semantics. Later, these view definitions transformed to the MOVE specific algorithms to extract quality, optimized sub-ontologies. Here, since the view representations are at a higher level of abstraction, the definitions can be mapped to other SW languages such as, RDF-S or OWL, if required. To the best of our knowledge, this is one of the early works in the SW domain that provides a view formalism with support for conceptual semantics with a compressive deployment solution architecture (i.e. the MOVE system).

2.7 Issues Associated with View Formalisms

In this section, in addition to the discussions presented in the above section, here we look at some common issues associated with the concept of views. These include view models that do not fall into the above categories, conceptual semantics, constraint specification for views and other issues related to the notion of views.

2.7.1 Alternative View Concepts

Here, we consider some of the alternative view concepts: Object-Relational (O-R) views and graph structured views and XML views for structured data.

There exists some work that explores the notion of views in the Object-Relational (O-R) paradigm. Object relational is an extended relational model with the aim of extending current relational databases technology to support OO concepts and while maintaining the performance and non-procedural features of the relational model. In theory, O-R model extends the relational systems by introducing the features of object references, type inheritance, collections, and aggregate functions including user-defined set aggregates, and nested row types (Rahayu 2000).

However, the view concept in the O-R model is similar to that of the traditional relational model with support for some complex types, as opposed to simple type definitions. In addition, in the related work, using such O-R view concept, some extended concepts, in the context of data warehousing has been developed in work such as in (Gopalkrishnan, Li & Karlapalem 1999). However, since O-R model semantics (and concepts) are closely related or similar to the relational model, in this research, we consider O-R views as a sub-set of relational views and do not elaborate further.

In the context of this thesis, graph structured data models, such as DOM (W3C-DOM 2004) base models, OEM base models (Zhuge & Garcia-Molina 1998), etc. are irrelevant to our research as: (a) they lack high-level data semantics and the concept of semantic relationships (as opposed to structural relationships); (b) provide no high-level design methodologies, such OO conceptual modeling; and (c) they are usually an extension or abstraction of a the implementation mechanism for other data models, such as semi-structured data. Though some graph structured data models provide some form of view formalism (Zhuge & Garcia-Molina 1998), due to the reasons stated (a) – (c) above, we do not elaborate further on such works.

It is imperative to differentiate one misunderstood notion of XML views: that is, the concept of so-called "XML views", promoted by many (mostly relational) database vendors, where structured data (relational, OO) stored in a database is extracted (using SQL or external program modules), encoded in XML markup tags and disseminated over the web (or application domains). Here, in this research, we do not consider such a notion as a view formalism for XML, as it is just another document with XML mark-ups without view properties or semantics.

2.7.2 Conceptual Semantics for Views

To the best of our knowledge, in the existing literature, the notion of *conceptual semantics* and *conceptual representation* for views are non-existent. However, some consideration for logical semantics in views are considered in (Volz, Oberle & Studer 2003b), this work being in the context of the SW paradigm. Conversely, there exist some works that claim to consider views as first-class citizens of the database schema (as opposed to the conceptual model). Here, at the database schema level which is usually considered as the logical level of a database, the model elements (e.g. views, relationships, etc.) are dependent on the data model against which the schema is valid. Therefore, it is difficult to represent unconstrained conceptual level semantics at this level.

Another research direction is in (Balsters 2003) that looks at the notion of relational views (i.e. named query) and proposed to model and specify such views using the concept of derived classes in UML (OMG-UML™ 2003a). For such a purpose, the research considers using the OMG's Object Constraint Language (OCL) (OMG-OCL 2003) as a representational language for specifying relational views. However, there are few drawbacks associated with this type of view specification, namely:

(i) the notion of view is still considered as a "named query" and UML/OCL only used a representation for the view query, which serves no purpose other than (view) query visualization;

(ii) OCL is declarative in nature, thus its textual representation is more complex and may include more ambiguity than the traditional SQL statements at the DBMS level;

(iii) although it is "represented" using UML/OCL, the view notion does not accommodate conceptual semantics and relationships such as hierarchical and/or constraints that are easily deployable. Also, views are not considered as first-class citizens in the UML/OCL representational model;

(iv) the view specification here requires in-depth knowledge of OCL (non-visual and textual in nature) and extensions to the OCL standards. In addition, the OCL is designed with OO concepts in mind and the research is focused on using it for specifying relational views which requires further transformation and/or mapping formalism to deploy such UML/OCL views to relational models.

Therefore, though the relational UML/OCL views (Balsters 2003) use UML/OCL only as a representational notation, thus, it is not considered as view formalism with conceptual semantics. Rather, it is an alternative, declarative method of describing relational views other than SQL.

2.7.3 View Constraints

In data modelling, specifications often involve constraints. In the case of views, it is usually specified by the data language in which they are defined. For example, in a relational model, views are defined using SQL and a limited set of constraints can be defined using SQL(Date 2003; Elmasri & Navathe 2004), namely:

- presentation specific (such as display headings, column width, pattern order etc);
- range and string patterns for aggregate fields;
- input formats for updatable views; and
- other DBMS specific (such view materialization, table block, size, caching options etc).

In Object-Relational and OO models, views had similar constraints but they are more extensive and explicit due to the data model. The views here are constructed and specified by DBMS specific (such as OQL(Cattell et al. 2000)) and/or external languages (such as C++, Java or O_2C(Abiteboul & Bonner 1991)). It is a similar situation in views for the semi-structured data paradigm, where a rich set of view constraints are defined using languages such as OQL based LOREL (Abiteboul, Quass, McHugh, Widom et al. 1997; Goldman, McHugh & Widom 1999). But the work by authors of (YB Chen, TW Ling & ML Lee 2002) provides some form of high-level view constraints (under ORA-SS model) for XML views, while the work in (Volz, Oberle & Studer 2003b) provides some form of logical level view constraints to be defined in views for in SW/RDF paradigm.

2.7.4 Remarks

The discussion of the existing view formalism presented above, especially with the views for XML, it raises many concerns in regards to views for XML (and semi-structured data including XML, OWL, RDF, etc.). They are:

(vii) conceptual semantics and views: From the work presented above, though useful, the notion of views are never separated from the implementation (and data instance) concerns. Thus, though very usual, all view formalism presented above did not utilizes conceptual semantics in defining and specifying views;

(viii) view specification and construction: View construction is always synonymous concept as part of the query language. To this date, for view construction, no mechanism that is independent of view language syntax semantics exists;

(ix) modelling and transformation: since the views are usually part of the external schema, for the purpose of modelling, the view semantics are never available at a higher level of abstraction. Thus the notion of "modelling" does not exist for view formalism;

(x) constraints and views: as stated before, since view constraints are coupled with language, view constraint specifications are not easily portable other than the intended model or language. This, in our understanding, causes problem especially with distributed models such as XML and Semantic Web.

From the discussions above, we find it essential to focus mainly on the highest level of abstraction, the conceptual level for specifying and defining views, in the context of XML.

2.8 Conclusion

This thesis aims to provide a solid foundation for an area that is virtually unexplored, that is, specifying and defining views for XML at the highest level of abstraction - higher than query languages. The main benefit of this approach is to capture and represent XML specific semantics at a high level abstraction that can be modeled, communicated and transformed to other required levels of abstraction without loss of semantics.

In this chapter we provided detailed discussion of existing view formalisms, their characteristics, and drawbacks in chronological order of existing data models, from relational to SW models. We also presented some of the desired characteristics and requirements for XML view formalism that is easily adaptable and extensible, with respect to the problems faced by the existing view formalisms.

In the following chapter, taking the findings of this chapter into consideration, we will formally define the problem addressed in this thesis, followed by a detailed discussion of the outline of the problem in Chapter 4.

References

Abiteboul, S 1999, 'On Views and XML', *Proceedings of the eighteenth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems (PODS '99)*, ACM Press New York, NY, USA, Philadelphia, Pennsylvania, USA, pp. 1-9.

Abiteboul, S, Amann, B, Cluet, S, Eyal, A, Mignet, L & Milo, T 1999, 'Active Views for Electronic Commerce', *Proceedings of the 25th International Conference on VLDB*, Edinburgh, Scotland, pp. 138-49.

Abiteboul, S, Benjelloun, O, Manolescu, I, Milo, T & Weber, R 2002a, 'Active XML: A Data-Centric Perspective on Web Services', *BDA*.

Abiteboul, S, Benjelloun, O, Manolescu, I, Milo, T & Weber, R 2002b, 'Active XML: Peer-to-Peer Data and Web Services Integration', *Proceedings of the 28th International Conference on VLDB*, HK, China.

Abiteboul, S, Benjelloun, O & Milo, T 2004, 'Positive Active XML', *Proceedings of PODS*, ACM, France.

Abiteboul, S & Bonner, A 1991, 'Objects and Views', *ACM SIGMOD Record, Proceedings of the International Conference on Management of Data (ACM SIGMOD '91)*, vol. 20 (2), ACM Press New York, NY, USA.

Abiteboul, S, Buneman, P & Suciu, D 1999, *Data on the Web : from relations to semistructured data and XML*, Morgan Kaufmann, London, UK.

Abiteboul, S, Goldman, R, McHugh, J, Vassalos, V & Zhuge, Y 1997, 'Views for Semistructured Data', *Workshop on Management of Semistructured Data*, USA.

Abiteboul, S, Quass, J, McHugh, J & Widom, J 1997, 'Lore: A Database Management System for Semistructured Data', *SIGMOD Record*, vol. 26(3), ACM, pp. 54-66.

Abiteboul, S, Quass, J, McHugh, J, Widom, J & Wiener, J 1997, 'The Lorel Query Language for Semistructured Data', *International Journal on Digital Libraries*, vol. 1, no. 1, pp. 68-88, April 1997.

Aguilera, V, Cluet, S, Milo, T, Veltri, P & Vodislav, D 2002, 'Views in a Large-Scale XML Repository', *The VLDB Journal —The International Journal on Very Large Data Bases*, vol. 11(3), no. 3, pp. 238-55

Alhajj, R, Polat, F & Yilmaz, C 2005, 'Views as Frist-Class Citizens in Object-Oriented Databases', *The VLDB Journal —The International Journal on Very Large Data Bases*, vol. 14(2), no. 2, pp. 155-69

ANSI & ISO 2003, *ANSI - SQL 2003*, ANSI / ISO.

Augurusa, E, Braga, D, Campi, A & Ceri, S 2003, 'Design and Implementation of a Graphical Interface to XQuery', *ACM Symposium on Applied Computing (SAC '03)*, ACM, Melbourne, USA, pp. 1163-7.

Balsters, H 2003, 'Modelling Database Views with Derived Classes in the UML/OCL-framework', *The Unified Modeling Language: Modeling Languages and Applications (UML '03)*, vol. 2863, Springer, USA, pp. 295-309.

Baru, C, Ludäscher, B, Papakonstantinou, Y, Velikhov, P & Vianu, V 1998, 'Features and Requirements for an XML View Definition Language: Lessons from XML Information Mediation', *W3C's Query Language Workshop (position paper)*.

Benjelloun, O 2004, 'Active XML: A data centric perspective on Web services', PhD thesis, Paris XI University, Orsay, France.

Bertino, E 1992, 'A View Mechanism for Object-Oriented Databases', *3rd International Conference on Extending Database Technology (EDBT '92)*, vol. 580, eds A Pirotte, C Delobel & G Gottlob, Springer, Vienna, Austria, pp. 136-51.

Braga, D & Campi, A 2003, 'A Graphical Environment to Query XML Data with XQuery', *4th International Conference on Web Information Systems Engineering (WISE '03)*, IEEE Computer Society, Rome, Italy, pp. 31-40.

Braga, D, Campi, A & Ceri, S 2005, 'XQBE (XQuery By Example): A visual interface to the standard XML query language', *ACM Transactions on Database Systems (TODS)*, vol. 30(2), no. 2, pp. 398-443

Cattell, RGG, Barry, DK, Berler, M, Eastman, J, Jordan, D, Russell, C, Schadow, O, Stanienda, T & Velez, F (eds) 2000, *The Object Data Standard: ODMG 3.0*, Morgan Kaufmann.

Ceravolo, P, Damiani, E, Elia, G & Viviani, M 2006, 'Bottom-up Extraction and Main-tenance of Ontology-based Metadata', in *Fuzzy Logic and the Semantic Web*, Elsevier.

Ceravolo, P, Damiani, E & Viviani, M 2005, 'Adding a Peer-to-Peer Trust Layer to Metadata Generators', *Proceedings of the OTM '05 Workshops*, pp. 809-15.

Chang, E 1996, 'Object Oriented User Interface Design and Usability Evaluation', Doctor of Philosophy (Ph.D) thesis, La Trobe University, Melbourne, Australia.

Chang, E & Dillon, TS 1994, 'Integration of User Interfaces with Application Software and Databases Through the Use of Perspectives', *1st International Conference on Object-Role Modeling (ORM '94)*, Australia, pp. 4-6.

Chang, E, Dillon, TS & Hussain, FK 2006, *Trust and Reputation for Service Oriented Environments: Technologies for Building Business Intelligence and Consumer Confidence*, John Wiley and Sons, UK.

Chen, PP 1976, 'The Entity-Relationship Model - Towards a Unified View of Data', *ACM Transaction on Database Systems (TODS)*, vol. 1(1), pp. 09-36

Chen, YB, Ling, TW & Lee, ML 2002, 'Designing Valid XML Views', *Proceedings of the 21st International Conference on Conceptual Modeling (ER '02)*, vol. 2503, eds S Spaccapietra, ST March & Y Kambayashi, Springer-Verlag London, UK, Tampere, Finland, pp. 463 - 78.

Chen, YB, Ling, TW & Lee, M-L 2002, 'A Case Tool for Designing XML Views', *Second International Workshop on Data Integration over the Web (DiWeb '02)*, ed. Ze Lacroix, University of Toronto Press, Toronto, Canada, pp. 47-57.

Cluet, S, Veltri, P & Vodislav, D 2001, 'Views in a Large Scale XML Repository', *Proceedings of the 27th VLDB Conference (VLDB '01)*, Roma, Italy.

Codd, EF 1970, 'A Relational Model of Data for Large Shared Data Banks', *Communications of the ACM*, vol. 13(6), no. 6, pp. 377-87

Codd, EF 1983, 'A relational model of data for large shared data banks', *Communications of the ACM*, vol. 26(1), no. 1, pp. 64-9

Codd, EF 1990, *The Relational Model for Database Management: Version 2*, Addison Wesley Publishing Company.

Date, CJ 2003, *An introduction to database systems*, 8th edn, Pearson/Addison Wesley, New York.

Dillon, TS & Tan, PL 1993, *Object-Oriented Conceptual Modeling*, Prentice Hall, Australia.

Elmasri, R & Navathe, S 2004, *Fundamentals of database systems*, 4th edn, Pearson/Addison Wesley, New York.

Feng, L, Chang, E & Dillon, TS 2002, 'A Semantic Network-based Design Methodology for XML Documents', *ACM Transactions on Information Systems (TOIS)*, vol. 20, No 4, no. 4, pp. 390 - 421

Feng, L, Chang, E & Dillon, TS 2003, 'Schemata Transformation of Object-Oriented Conceptual Models to XML', *International Journal of Computer Systems Science & Engineering*, vol. 18, No. 1, no. 1, pp. 45-60

Gâomez-Pâerez, A, Fernâandez-Lâopez, M & Corcho, O 2003, *Ontological engineering : with examples from the areas of knowledge management, e-commerce and the semantic Web*, Advanced information and knowledge processing., Springer-Verlag, London ; New York.

Gaševic, D, Djuric, D & Devedzic, V 2005, 'Bridging MDA and OWL Ontologies', *Journal of Web Engineering*, vol. 4(2), no. 2, pp. 118-43

Gaševic, D, Djuric, D, Devedzic, V & Damjanovic, V 2004a, 'Approaching OWL and MDA Through Technological Spaces', *3rd Workshop in Software Model Engineering (WiSME 2004)*, Lisbon, Portugal.

Gaševic, D, Djuric, D, Devedzic, V & Damjanovic, V 2004b, 'Converting UML to OWL Ontologies', *Proceedings of the 13 th International World Wide Web Conference*, NY, USA, pp. 488-9.

Gaševic, D, Djuric, D, Devedžic, V & Damjanovic, V 2004, 'UML for Read-To-Use OWL Ontologies', *Proceedings of the IEEE International Conference Intelligent Systems*, Vrana, Bulgaria.

Goldman, R, McHugh, J & Widom, J 1999, 'From Semistructured Data to XML: Migrating the Lore Data Model and Query Language', *Proceedings of the 2nd International Workshop on the Web and Databases (WebDB '99)*, Philadelphia, Pennsylvania.

Gopalkrishnan, V, Li, Q & Karlapalem, K 1999, 'Star/Snow-flake Schema Driven Object-Relational Data Warehouse Design and Query Processing Strategies', *1st First International Conference on Data Warehousing and Knowledge Discovery (DaWaK '99)*, Springer, Florence Italy.

Hadzic, M & Chang, E 2004, 'Role of the Ontologies in the Context of Grid Computing and Application for the Human Disease Studies', *Semantics for Grid Databases, First International IFIP Conference on Semantics of a Networked World (ICSNW '04)*, vol. LNCS 3226, Springer Verlag, Paris, France, pp. 316-8.

HyOntUse 2003, *User Oriented Hybrid Ontology Development Environments, (http://www.cs.man.ac.uk/mig/projects/current/hyontuse/)*.

Jarrar, M, Demey, J & Meersman, R 2003, 'On Using Conceptual Data Modeling for Ontology Engineering', *Journal on Data Semantics*, vol. LNCS 2800, pp. 185-207

KAON 2004, *KAON Project (http://kaon.semanticweb.org/Members/rvo/Folder.2002-08-22.1409/Module.2002-08-22.1426/view)*, 2005.

Kim, W 1990, 'Research Directions in Object-Oriented Database Systems', *Proceedings of the ninth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, ACM Press New York, NY, USA, Nashville, Tennessee, USA, pp. 1-15.

Kim, W & Kelly, W 1995, 'Chapter 6: On View Support in Object-Oriented Database Systems', in *Modern Database Systems*, Addison-Wesley Publishing Company, pp. 108-29.

Kuno, H & Rundensteiner, E 1993, 'Developing an Object-Oriented View Management System', *Proceedings of the Conference of the Centre for Advanced Studies on Collaborative research: software engineering*, vol. 1, IBM Press, Toronto, Ontario, Canada, pp. 548 - 62.

Lucie-Xyleme 2001, 'Xyleme: A Dynamic Warehouse for XML Data of the Web', *International Database Engineering & Applications Symposium (IDEAS '01)*, eds ME Adiba, C Collet & BC Desai, IEEE Computer Society 2001, Grenoble, France, pp. 3-7.

Ludaescher, B, Papakonstantinou, Y & Velikhov, P 2000, 'Navigation-Driven Evaluation of Virtual Mediated Views', *Extending DataBase Technology (EDBT '00)*, Springer.

Ludaescher, B, Papakonstantinou, Y, Velikhov, P & Vianu, V 1999, 'View Definition and DTD Inference for XML', *Post-ICDT Workshop on Query Processing for Semistructured Data and Non-Standard Data Formats.*

Munroe, K & Papakonstantinou, Y 2000, 'BBQ: A Visual Interface for Integrated Browsing and Querying of XML', *Visual Database Systems (VDB '00).*

Noy, N & Musen, M 2002, 'Promptdiff: a fixed-point algorithm for comparing ontology versions', *Proceedings of the Eighteenth National Conference on Artificial Intelligence*, American Association for Artificial Intelligence, pp. 744-50.

OMG-OCL 2003, *UML 2.0 OCL Final Adopted specification (http://www.omg.org/cgi-bin/doc?ptc/2003-10-14)*, OMG, 2005.

OMG-ODM 2005, *Ontology PSIG, (http://www.omg.org/ontology/)*, OMG, 2006.

OMG-UML™ 2003a, *UML 2.0 Final Adopted Specification (http://www.uml.org/#UML2.0)*, OMG, 2005.

OMG-UML™ 2003b, *Unified Modeling Language™ (UML) Version 1.5 Specification*, OMG.

Rahayu, JW 2000, 'Object-Relational Transformation Methodology', Doctor of Philosophy (Ph.D) thesis, La Trobe University, Melbourne, Australia.

Spyns, P, Meersman, R & Mustafa, J 2002, 'Data Modeling Versus Ontology Engineering', *SIGMOD*, pp. 14-9.

Tsichritzis, D & Klug, A 1978, 'The ANSI/X3/SPARC/DBMS Framework Report of the Study Group on Database Management Systems', *Information Systems*, vol. 3(3), no. 3, pp. 173-91

Uceda-Sosa, R, Chen, CX & Claypool, KT 2004, 'CLOVE: A Framework to Design Ontology Views', *23th International Conference on Conceptual Modeling (ER '05)*, Springer-Verlag, Shanghai, China, pp. 844-9.

Ullman, JD 1989, *Principles of database and knowledge-base systems*, vol. 1, Principles of computer science series., Computer Science Press, Rockville, Md.

Volz, R, Oberle, D & Studer, R 2003a, 'Implementing Views for Light-Weight Web Ontologies', *Seventh International Database Engineering and Applications Symposium (IDEAS'03)*, IEEE Computer Society, Hong Kong, SAR, pp. 160-.

Volz, R, Oberle, D & Studer, R 2003b, 'Views for light-weight Web ontologies', *Proceedings of the ACM Symposium on Applied Computing (SAC '03)*, ACM Press New York, NY, USA, USA, pp. 1168 - 73.

W3C-DOM 2004, *Document Object Model (DOM), (http://www.w3.org/DOM/)*, W3C, 2005.

W3C-DTD 2001, *XML Schema (http://www.w3.org/*, W3C, 2005.

W3C-OWL 2004, *OWL: Web Ontology Language 1.0 reference (http://www.w3.org/2004/OWL/)*, W3C, viewed 2005.

W3C-RDF 2004, *Resource Description Framework (RDF), (http://www.w3.org/RDF/)*, The World Wide Web Consortium (W3C), viewed March 2000.

W3C-RDFS 2004, *RDF-Schema (RDF-S), (http://www.w3.org/TR/rdf-schema/)*, The World Wide Web Consortium (W3C), viewed March 2000.

W3C-RDQL 2004, *RDQL - A Query Language for RDF, (http://www.w3.org/Submission/2004/SUBM-RDQL-20040109/)*, W3C.

W3C-SW 2005, *The Semantic Web (http://www.w3.org/2001/sw/)*, W3C.

W3C-XML 2004, *Extensible Markup Language (XML) 1.0, (http://www.w3.org/XML/)*, The World Wide Web Consortium (W3C).

W3C-XPath 1999, *XML Path Language (XPath) Version 1.0 (http://www.w3.org/TR/xpath/)*, The World Wide Web Consortium (W3C), November 1999.

W3C-XQuery 2004, *XQuery 1.0: An XML Query Language (http://www.w3.org/TR/xquery)*, The World Wide Web Consortium (W3C), viewed November 2003.

W3C-XSD 2001, *XML Schema (http://www.w3.org/XML/Schema)*, W3C, 2004.

Wollersheim, D & Rahayu, JW 2005, 'Ontology Based Query Expansion Framework for Use in Medical Information Systems', *International Journal of Web Information Systems*, vol. 1, no. 2, pp. 101-15

Wongthamtham, P, Chang, E & Dillon, TS 2004, 'Methodology for Multi-site Software Engineering Using Ontology', *Proceedings of the International Conference on Software Engineering Research and Practice (SERP '04)*, CSREA Press, pp. 477-82.

Wouters, C 2006, 'A Formalization and Application of Ontology Extraction', Doctor of Philosophy (Ph.D) thesis, La Trobe University, Melbourne, Australia, Melbourne.

Wouters, C, Dillon, TS, Rahayu, JW & Chang, E 2002, 'A Practical Walkthrough of the Ontology Derivation Rules', *Database and Expert Systems Applications : 13th International Conference (DEXA '02)*, Springer, Aix-en-Provence, France, pp. 259-68.

Wouters, C, Dillon, TS, Rahayu, JW & Chang, E 2005, 'Large scale ontology visualisation using ontology extraction', *International Journal of Web and Grid Services*, vol. 1(1), no. 1, pp. 113 - 35

Wouters, C, Dillon, TS, Rahayu, JW, Chang, E & Meersman, R 2004a, 'Ontologies on the MOVE', *9th International Conference on Database Systems for Advanced Applications (DASFAA '04)*, vol. 2973/2004, eds Y Lee, J Li, K-Y Whang & D Lee, Springer-Verlag GmbH, Jeju Island, Korea, pp. 812-23.

Wouters, C, Dillon, TS, Rahayu, JW, Chang, E & Meersman, R 2004b, 'A Practical Approach to the Derivation of a Materialized Ontology View', in D Taniar & W Rahayu (eds), *Web Information Systems*, Idea Group Publishing, USA.

Xyleme 2001, *Xyleme Project (http://www.xyleme.com/)*, http://www.xyleme.com/.

Zhuge, Y & Garcia-Molina, H 1998, 'Graph structured Views and Incremental Maintenance', *Proceeding of the 14th IEEE Conference on Data Engineering (ICDE '98)*, IEEE, USA.

Chapter 3

Problem Formulation

3.1 Introduction

In Chapter 2, existing view models and techniques for defining and constructing such views were investigated and evaluated. Since XML is a relatively new technology, a number of shortcomings and problems were identified in the existing XML view models (and XML-enabled view models).

This chapter describes the problem being addressed in this thesis. In order to do this, it is imperative that one clearly understands and defines (a) what exactly is meant by the notion of *view* in the context of XML; and (b) an in-depth understanding of XML technologies and the notion of view-driven e-solutions.

This chapter is organized into three parts. The first part (sections 3.2-3.4) provides some background information on XML technologies. This includes sections 3.2 which discuss XML as a data model, followed by section 3.3 that provide general introduction to views. The second part of this chapter addresses the challenges and requirements of a view model for XML, in sections 3.6 and 3.7. It provides a detailed discussion of core concepts and foundations of the problem addressed in thesis. The third part looks at the problem itself. Section 3.8, describes the overview of the problem, while section 3.10 provides some feasible alternatives that are possible solutions to the problem addressed in this thesis. Section 3.11 summarizes the problem formulation and concludes this chapter.

3.2 Object-Oriented (OO) Concepts

In this section, we look at some of the OO concepts, notations and definitions used in this thesis.

3.2.1 Universe of Discourse

Universe of Discourse (UoD) is a familiar concept in logic, linguistics, and mathematics and computational theory. UoD develops an abstraction of the real world that captures relevant properties, characteristics, relationships, constraints and knowledge. Further, UoD is the set of all objects presumed or hypothesized to exist for some specific purpose. Objects may be concrete or abstract. Objects may be primitive or composite. Also, all the sets of entities defined in the UoD are capable of being represented in some declarative language or other formal systems.

Definition 3.1. Universe of Discourse (UoD) is an abstraction of the part of the world that is under discussion and consists of a set of descriptors that imply different objects, their properties, their relationships and constraints.

For the purpose of this thesis, let us denote the UoD as UoD. Also, let O_{bject} be a set of descriptors that uniquely identify *objects* in the real world, $A_{ttribute}$ be a set of descriptors that uniquely identify *attributes* of these objects in the real world, M_{ethod} be a set of descriptors that uniquely identify *methods* in the real world which represent the behavioural properties of an object, $R_{elationship}$ be a set of descriptors that uniquely identify *relationships* between a set of objects in the real world and $C_{onstraint}$ be a set of descriptors that uniquely identify constraints associated with O_{bject}, $A_{ttribute}$, M_{ethod} and $R_{elationship}$ in the real world.

Definition 3.2. The Universe of Discourse $\bigcup_{UoD}$ is a set of descriptors uniquely identifying objects O_{bject}, attributes $A_{ttribute}$, methods M_{ethod}, relationships $R_{elationship}$ and constraints $C_{onstraint}$ in the real world.

In our choice of descriptors for an element of object, we have chosen to use an object that conforms to the concept of an object in an Object-Oriented system.

An object in the OO model encapsulates the structural as well as behavioural aspects of a real world object. The structural aspects consist of the objects, attributes, methods and the semantic (and structural) relationships between them, namely, inheritance, association, and aggregation. Each of these relationships is associated with a set of constraints. The dynamic aspect of the object-oriented conceptual model is divided into two types, namely *generic* methods and *user-defined* methods.

$$UoD = \{O_{bject} \cup A_{ttribute} \cup M_{ethod} \cup R_{elationship} \cup C_{onstraint}\} \qquad (3.1)$$

Thus, it can be also stated as: an Object-Oriented (OO) conceptual model represents a real-world domain at a required level of abstraction and granularity, using a collection of objects, entities, relationships and constraints. It provides the representation of the domain for software modules and database schemas to be built. There are several ways of formally representing classes in an OO model. For example, in works such as (Chang 1996; T.S. Dillon & Tan 1993; Graham, Wills & O'Callaghan 2001; Rahayu 2000), authors have shown different ways of formally representing classes, relationships and concepts that are capable of representing real-world properties. The definitions and the notations used in this thesis are given below.

3.2.2 Objects

Let x^i_{obj} denotes an object (where $x^i_{obj} \in O_{bject}$). We represent a collection of objects with similar properties as X_{obj} and hence we write:

$$X_{obj} = \{x^1_{obj}, x^2_{obj}, ..., x^i_{obj}, ..., x^n_{obj}\} \qquad \textbf{(3.2)}$$

where $1 \leq i \leq n$.

$$\forall x^i_{obj} \in X_{obj} : x^i_{obj} = (y) \ \ with \ \ y \in O_{bject} \qquad \textbf{(3.3)}$$

Since x^i_{obj} corresponds to a single instance of an object in the real world, we refer to x^i_{obj} as an instance object.

3.2.3 Attributes

Static properties of an object are described by its attributes and attribute constraints.

Definition 3.3. Attributes are the data or variables that characterize the state of an object (T.S. Dillon & Tan 1993).

Let A_X be a set of attributes be of the object X_{obj}.

$$A_X = \{a^1_X, a^2_X, ..., a^j_X, ..., a^m_X\} \qquad \textbf{(3.4)}$$

where $1 \leq j \leq m$.

$$\forall a_X^j \in A_X : a_X^j = (y) \;\; with \;\; y \in A_{ttribute} \qquad (3.5)$$

3.2.4 Methods

Definition 3.4. Methods are the actions or procedures that change the state of an object (T.S. Dillon & Tan 1993).

Let M_X be a set of methods of the object X_{obj}.

$$M_X = \{m_X^1, m_X^2, ..., m_X^k, ..., m_X^o\} \qquad (3.6)$$

where $1 \le k \le o$.

$$\forall m_X^k \in M_X : m_X^k = (y) \;\; with \;\; y \in M_{ethod} \qquad (3.7)$$

3.2.5 Constraints

Definition 3.5. A constraint is a restriction or an extended description of one or more model elements (or artifacts) and their values.

Definition 3.6. A *Con* constraint is a set of two-tuples with the property $c_{property}$ and constraint expression $c_{expressison}$.

$$Con = \{(c_{property}, c_{expressison})^1, \ldots\ldots.(c_{property}, c_{expressison})^q, \ldots\ldots\ldots.\} \qquad (3.8)$$

where q is a finite integer .

$$\forall (p, exp) \in Con : \{(p, exp)\} = \{(y_1, y_2)\} \ \ with \ (y_1, y_2) \in C_{onstraint} \qquad (3.9)$$

Let Con_X be a set of constraints (where $Con_X \in C_{onstraint}$) associated with X_{obj}, A_X and M_X . Thus:

$$Con_X = \{con_X^1, con_X^2, \ldots., con_X^p, \ldots., con_X^o\} \qquad (3.10)$$

where $1 \leq p \leq o$.

$$\forall con_X^p \in Con_X : con_X^p = (y) \ \ with \ y \in C_{onstraint} \qquad (3.11)$$

3.2.6 Object Schema

An *object schema* may be considered as a collection of descriptions and definitions of properties of a given concept or notion in a systematic manner using a language syntax or notation (usually textual) that can be easily interpreted by some computer program (or human reader).

Definition 3.7. An object schema S_X defined as a triple composing a set of attributes A_X , a set of methods M_X and a set of constraints Con_X associated with attributes and methods.

$$S_X = (A_X, M_X, Con_X) \qquad (3.12)$$

Form equations 3.5, 3.7 and 3.9, it can be shown that:

$$\forall s_X \in S_X : s_X = (y) \ \textit{with} \ y \in UoD \qquad (3.13)$$

3.2.7 Class

A class is the basic notion of abstraction (T.S. Dillon & Tan 1993) in OO models. A class is composed of both attributes (static properties) and methods (dynamic properties). These attributes and methods define the states and behaviors associated with a class. There are several ways of formally representing classes in an OO model. For example, in works such as (Chang 1996; T.S. Dillon & Tan 1993; Graham, Wills & O'Callaghan 2001; Rahayu 2000), authors have shown different ways of formally representing a class. In this thesis, we follow an extended notation that is based on (Rahayu 2000), as shown below.

Definition 3.8. A class is a set of objects with common properties that are classified or grouped together and have a common object schema.

Let C_{lass}^{X} denote a class. Thus more formally a class in OO is defined as;

Definition 3.9. A class C_{lass}^{X} is a two tuple, having a set of objects X_{obj} and a schema S_X, such that, S_X describes the X_{obj} properties, behavior and constraints of the class.

$$C^{X}_{lass} = (S_X, X_{obj}) \qquad \textbf{(3.14)}$$

That is to say, a class is a blueprint for creating new instance objects of similar properties and behaviours. Thus, a class is considered to be an *abstract level* entity of a set of *instance level* objects similar properties and behaviours (T.S. Dillon & Tan 1993).

Definition 3.10. A class is an *abstract* level entity of generic category of a set of *instance* level objects with similar properties and behaviour.

3.2.8 Class Vs. Types

For the purpose of this work, it is important to clarify the notion of types and classes. In the OO community, there are some who distinguish a class from a type. A type can be described as (Tharam S. Dillon et al. 2006 (in pint); T.S. Dillon & Tan 1993);

- It corresponds to a notion of Abstract Data Type (ADT) and in programming languages variables are declared or defined using such types.
- Some examples of typed based languages include C++ (Stroustrup et al. '86), O2 (Atkinson et al. '89), SIMULA 67 (Dahl '68) etc. In these languages, the type definition of the variables indicates to the system, the structure of the type and the operations that are allowed on the values of the variables at run time. Also, variable type checking is done at the compile time to ensure program correctness and do not change during the execution (or runtime) of the program.
- Types are referred during the compile time and not during runtime, thereby ensuring program safety and runtime efficiency, but reducing program flexibility and user-friendliness.

Conversely, a class can be described as (Tharam S. Dillon et al. 2006 (in pint); T.S. Dillon & Tan 1993);

- A class consists of data structure and operations that are applicable to the class describing the states and behaviour that may be associated with its objects
- A blueprint for creating new instance objects
- Some examples of class based languages include SMALLTALK (Goldberg '83), Java (Sun Microsystems '96) Gemstone (Maier et al. '86). Unlike type based languages, a class is referred during runtime than during compile time, thus providing programming flexibility, uniformity and user-friendliness at the cost of runtime efficiency.

It is apparent that, in the context of programming, there are differences between a class-based language and a type-based language. But in the context of modelling (Tharam S. Dillon et al. 2006 (in pint); T.S. Dillon & Tan 1993; Graham, Wills & O'Callaghan 2001) (such as in our work), there are more similarities than distinctions between the two. In OO modelling, both classes and types are notationally identical (in almost all OO modelling languages) and are capable of being a blueprint for creating, manipulating and grouping objects. Though semi-structured data and their associated schemas (e.g. XML) exhibit strong characteristics of type-based systems in the context of programming, in the context of modelling, such differences do not matter as, at a higher level of abstraction, semi-structured data models exhibit strong similarities to both types and classes. Therefore, in this research, we do not distinguish types and classes.

3.2.9 Domain

Let $dT_1,.....,dT_n$ be a series of basic data types corresponding to the attribute set A_X, whose domains are denoted by $dom(dT)_1,.....,dom(dT_n)$.

Definition 3.11. A domain of a class is defined as equivalent to the union of all data type values permitted by its attribute set.

Let $dom(X)$ denote a domain to which the class X_{class} belongs.

$$dom(X) = \bigcup_{i=1}^{n} dom(dT_i) \qquad \textbf{(3.15)}$$

3.2.10 Relationships

Classes and objects in OO models have additional structural aspects that consist of the classes/objects and the structural relationships between them and the set of constraints associated with each relationship. Based on their semantic meaning and unique characteristics (in OO), relationships are classified into four types, namely: (a) inheritance ($Rel_{is\text{-}a}$), (b) association ($Rel_{association}$), (c) aggregation ($Rel_{aggregation}$) and (d) dependency ($Rel_{dependency}$). Thus, for the purpose of this thesis, in equation 3.1, $R_{elationship}$ can be stated as:

$$R_{elationship} = Rel_{is\text{-}a} \cup Rel_{association} \cup Rel_{aggregation} \cup Rel_{dependency} \qquad \textbf{(3.16)}$$

Definition 3.12. The Degree of a Relationship (DoR) is defined as the number of distinct object sets associated with the relationship in question.

Let Rel^{degree} denote the DoR Rel.

- If $Rel^{degree} = 1$, Rel is referred red to as a unary relationship
- If $Rel^{degree} = 2$, Rel is referred to as a binary relationship

- If $Rel^{degree} > 2$, Rel is referred to as n-ary relationship

It should be noted that, in this thesis, we are mainly concerned with binary relationships. Unless explicitly stated, all relationships should be considered as binary relationships.

Also, in this research we consider two categories of relationships, namely (T.S. Dillon & Tan 1993): (a) instance (for object / tuple) level relationships and (b) abstract level (for class/entity) level relationships.

(1) Instance (for object / tuple) level relationships

Definition 3.13. An *instance* level relationship is a distinctive connection descriptor between two or more instance level entities, such as objects and the constraint values associated with it.

Here, the instance level objects (for tuples) inherit the parent class's attributes, methods, default values and constraints.

(2) Abstract level (or class/entity) level relationships

Definition 3.14. An *abstract* level relationship is a distinctive connection descriptor between two or more abstract level entities, such as classes.

It should be noted that, an instance level relationship is an instance of an abstract level relationship. One abstract level relationship is a set that may have one or more instance relationships as members.

Let Rel $(Rel \in R_{elationship})$ be an abstract level relationship between two classes C^X_{lass}, C^Y_{lass}. Also, let $r_{x^p_{obj}, y^q_{obj}}$ be an instance level relationship between two distinct instance objects x^p_{obj} $(\in X_{obj})$ and y^q_{obj} $(\in Y_{obj})$, of classes C^X_{lass} and C^Y_{lass} respectively. Thus we can write Rel :

$$(C^X_{lass}, Rel, Con_{rel}, C^Y_{lass}) \qquad \textbf{(3.17)}$$

where, Con_{rel} is a set of constraints associated with the relationship Rel, such that:

$$\forall con_{rel} \in Con_{rel} : con_{rel} = (y) \quad with \quad y \in C_{constraint} \qquad \textbf{(3.18)}$$

and

$$Con_{rel} \neq Con_X \text{ or } Con_{rel} \neq Con_Y \qquad \textbf{(3.19)}$$

$$\forall rel \in Rel : rel = (y) \quad with \quad y \in R_{elationship} \qquad \textbf{(3.20)}$$

We can also show that the instance relationship $r_{x^p_{obj}, y^q_{obj}}$ as;

$$(x^p_{obj}, r_{x^p_{obj}, y^q_{obj}}, (cp, cv), y^q_{obj}) \qquad \textbf{(3.21)}$$

where, (cp, cv) is the constraint name and value associated with the relationship $r_{x^p_{obj}, y^q_{obj}}$

$$r_{x^p_{obj}, y^q_{obj}} \in Rel \text{ and } (cp, cv) \in Con_{Rel} \tag{3.22}$$

It should be noted that, $r_{x^p_{obj}, y^q_{obj}}$ is one of the members or instances of the set Rel.

Furthermore, relationship Rel may be classified in detail depending on the hierarchy and/or constraints (e.g. cardinality constraints, dependency constraints, etc.) that are defined over them. These include: (i) union inheritance, (ii) mutual exclusion inheritance, (iii) partition inheritance, (iv) multiple inheritance, (v) existence-dependent composition, (vi) existence-independent composition, (vii) exclusive composition, (viii) non-exclusive composition, (ix) ordered composition, (x) homogenous composition, (xi) strong adhesion and (xii) weak adhesion.

3.3 Object-Oriented Modeling Languages

In this thesis, to conceptually model and represent real-world semantics, we use the OO modeling paradigm (i.e. steps, notations and techniques). The OO conceptual modeling (OOCM) approach is a process of developing a high-level model of the problem one is intending solve. OOCM have been successfully applied to conventional software and software development, database applications, knowledge based system, etc, in developing successful solutions to complex software engineering problems (Booch 1993; T.S. Dillon & Tan 1993; Graham, Wills & O'Callaghan 2001). Simply, it is a collection of notations, tools and techniques to visually develop a representation of the real-world scenario suitable for producing a blueprint of the system that is precise, manageable, easily understood, and easily deployed into a working solution. Also, OOCM is comprehensively addressed in the literature and we closely follow the work of Dillon & Tan (T.S. Dillon & Tan 1993), in modeling and designing software concepts and systems.

It should be noted that one of the strongest motivation in conceptual modeling (e.g. ER-D, OOCM, DFD, etc.) is the visual representation of real-world semantics, captured in some form or agreed standard, so that one can easily interpret the model artifacts. For such a purpose, in this thesis, we use two graphical notations, namely: (a) eXtensible Semantic (XSemantic) nets (Feng, Chang & Dillon 2002) and (b) OMG's UML/OCL (OMG-OCL 2003; OMG-UML™ 2003) combination. These two notations are briefly outlined below (sections 3.3.1 & 3.3.2), and elaborated further in the later chapters.

3.3.1 XSemantic nets Notation

The eXtensible Semantic (XSemantic) net notation (formally known as semantic network model) was initially proposed by Feng et al. in (Feng, Chang & Dillon 2002) and has been extended in this thesis. The name change was the reflection of:

- the new extensions that are being proposed in thesis to support modeling of views and the Semantic Web;
- the confusion that may arise between the semantic network model in (Feng, Chang & Dillon 2002) (and the one used in thesis) and the semantic network models discussed in the literature particularly those commonly used in the Artificial Intelligent (AI) communities.

In this section, we briefly look at some of the basic properties of the original semantic network concepts and notations, as presented in (Feng, Chang & Dillon 2002). From this point onwards, we refer to the work of (Feng, Chang & Dillon 2002) as XSemantic nets.

The XSemantic net-based view design methodology comprises of two design levels: (a) semantic level and (b) schema level. The aim is to harness the conceptual modeling power of semi-structured data (namely XML) in order to narrow the gap between real-world objects and the XML document structures (Feng, Chang & Dillon 2002).

The first level corresponds to the Object-Oriented (OO) conceptual level and comprises two models, namely, the XML domain and the XML view models. This level is based on a modified semantic network (Feng, Chang & Dillon 2002) that provides semantic modeling of the XML domains.

The second level of the proposed methodology is concerned with detailed XML schema design for both domain and view objects defined at the semantic level, including *element/attribute declarations* and *simple/complex type definitions*. The mapping between these two design levels are the schemata transformation proposal stated in (Feng, Chang & Dillon 2002), and used to transform the semantic models into the XML Schema, based on which, XML documents can be systematically created, managed, and validated.

The original "modified" semantic network was a modified semantic net notation, to model XML domains using Object-Oriented conceptual modeling principles. The modification includes:

- Alternate representations of cycles in the original semantic network concept. In a semantic network, cycles may be present, connecting concepts and relationships. For e.g. if we are to model a simple Conference Publication System (CPS) using semantic network notation, PAPER have one or more AUTHOR/(s) and AUTHOR may have other PAPER/(s). Since XML is tree structured, such cyclic notions have to be resolved in the semantic network to maintain structural and constructional modeling similarities to XML.
- The addition of clusters to realistically capture real-world objects and their properties or descriptions (i.e. attributes constraints etc.). There are three clusters defined in the "modified" semantic network model, namely: (i) connection, (ii) connection cluster and (iii) connection cluster set (Feng, Chang & Dillon 2002). By doing so, it alerts the designers to differentiate between the different levels of complex and simple nodes

that are used to represent real-world *objects* (e.g. PAPER) and their *properties* (e.g. PaperID, Title etc.).

The "modified" semantic network provides semantic modeling of XML domains through four major components:

- a set of *atomic* and *complex* nodes, representing real-world object and classes;
- a set of *directed* labeled edges, representing *semantic relationships* between the objects and classes;
- a set of labels denoting different types of *semantic* relationships, including *aggregation, generalization, association*, and *of-property* relationships;
- a set of constraints defined over nodes and edges to constrain semantic relationships and object domains.

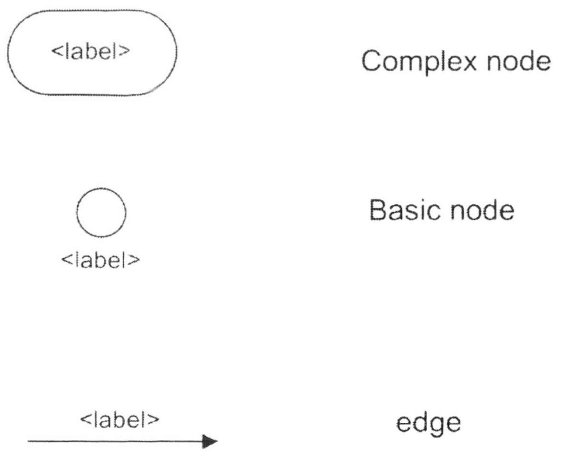


Figure 3.1: XSemantic net notations

The class in OO is represented as a complex node using the XSemantic net notation (Fig. 3.1). The class attributes of basic types are represented using atomic nodes. The directed edges represent OO semantic relationships; (a) IS-A denoted by the label "g", (b) aggregation/composition denoted by the label "a", (c) association denoted by the label "a" and (d) XSemantic net specific of-property relationship, denoted by the label "p" is used to graphically show the relationship (and the cardinality constraints) between a class and its attributes. Further more, additional constraints may be defined: (a) over a node, (b) over an edge, (c) over a set of nodes, and (d) over a set of edges. The basic XSemantic net notations are shown in Fig. 3.2.

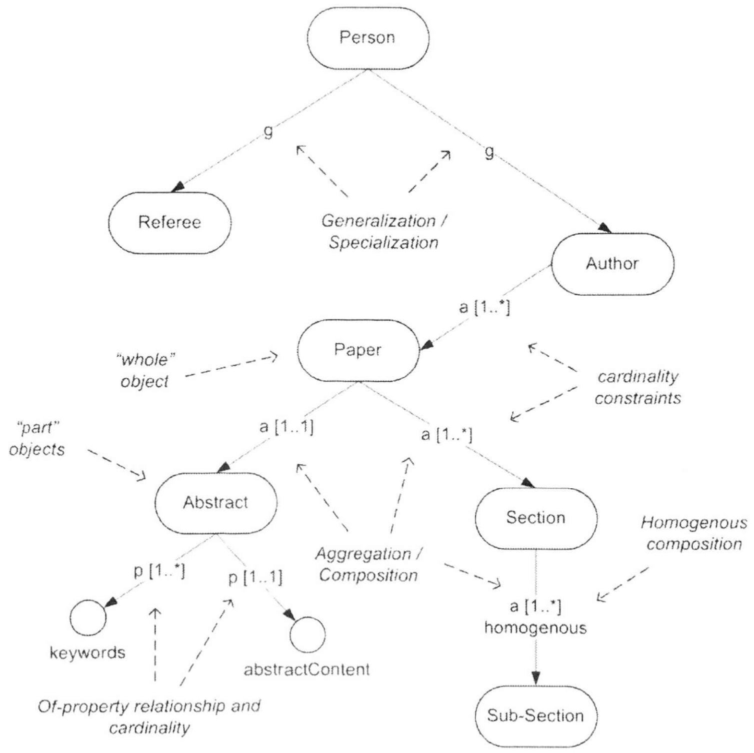


Figure 3.2: XSemantic net examples

3.3.2 Unified Modeling Language

The Unified Modeling Language (UML) is the result of an OMG initiative to standardize the graphical modeling notations for OOCM. At present, it the *defacto* modeling notation for OOCM among the software developer communities, and widely discussed, adapted and used as one of the most complete set of graphical notations that is available to date for OOCM. The parent language of UML, from which it was derived, is the Meta Object Facility (MOF) (OMG-MOF™ 2003).

Since all real-world concepts cannot to be depicted using graphical notation (e.g. data values, restrictions etc.), OMG supplemented the graphical notations with a textual descriptive predicate language call Object Constraint Language (OCL). The current version of UML, i.e. version 2, includes OCL as part of its modeling notation to promote the MDA (OMG-MDA 2003) modeling paradigm.

To represent real world objects, UML uses three main notations, namely class, relationship and constraints, as shown in Fig. 3.3. A class is a representation of real world objects, shown in UML using a rectangle with three containers, depicting: (a) a unique class name, (b) set of attributes describing the object's static properties, (c) a set of methods describing the object's behavior (or dynamic properties) and (d) visibility of the attributes and methods; that is, they are public, protected, private or derived.

Object relationships in UML are shown in UML using lines between classes, the shape and start-end points of the line depicting three main relationships in OO, namely; (a) generalization/specialization (or IS-A), as shown in Fig. 3.4(a), (b) aggregation (or part-of) as shown in Fig. 3.4(b), and (c) association (Fig. 3.4(c)). Relationship labels, cardinality and other constraints may be shown over the lines to describe and refine the relationships further. It should be noted that, *directed* relationships are not strongly emphasized in UML (Graham, Wills & O'Callaghan 2001; Rumbaugh, Booch & Jacobson 2004), but we consider it to be a useful and essential semantics associated with relationships. Thus, in this thesis, we consider all relationships to be directional (Graham, Wills & O'Callaghan 2001). That is, each relationship has a *start* and *end* (shown by an arrow head) point, as shown in Fig. 3.4. Bi-directional relationships are only allowed in special circumstances.

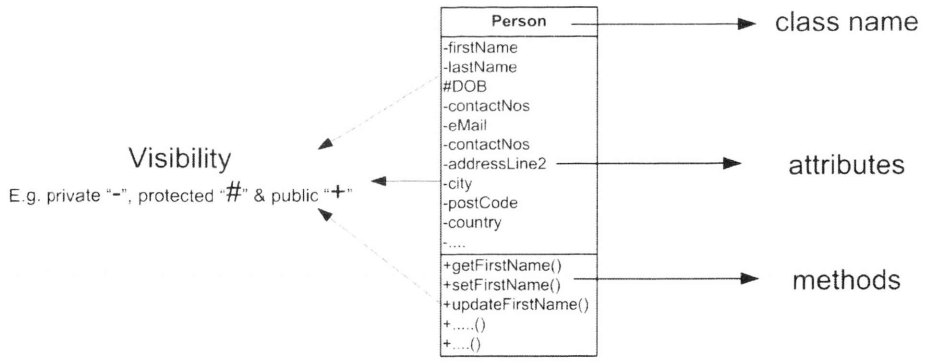


Figure 3.3: Graphical representation of class, attributes, methods and visibility in UML

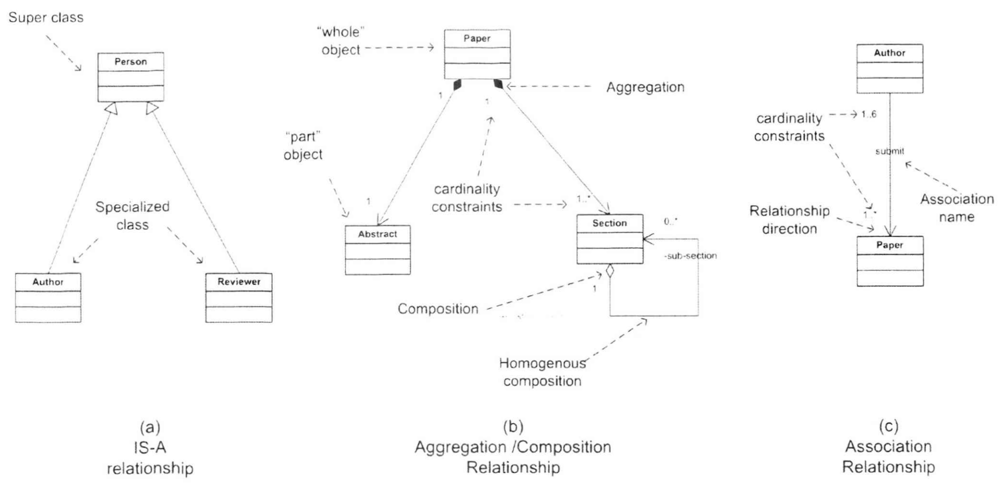


Figure 3.4: Graphical representation of OO relationships in UML

3.3.3 Transformation Function

A transformation function is function of a concept that maps the original concept representation or notation into another, based on some transformation rules or grammar. In this research, we denote a transformation function as:

$$\aleph_a^b(x) : x^a \rightarrow x^b \qquad (3.23)$$

where, x^a is the source and x^b is the transformed result.

For example, the schemata transformation described in the work (Feng, Chang & Dillon 2003), is a transformation function that transforms a UML model into XML Schema. Let m^{uml} denote a model in UML (source) and m^{xsd} a model in XML Schema (target). The schemata transformation between the models can be written as:

$$\aleph_{uml}^{xsd}(m) : m^{uml} \rightarrow m^{xsd}$$

3.4 Extensible Markup Language (XML)

Extensible Markup Language (XML) (W3C-XML 2004) was adopted as a W3C recommendation XML in 1998, which is an application profile of restricted form of the Standard Generalized Markup Language (SGML, ISO 8879), for electronic data representation and dissemination, in the context of electronic business, authoring, publication and exchange over the World Wide Web. Since its introduction, it has had rapidly evolved into a standard markup language for representing both data-centric and document-centric data/documents in unstructured (for e.g. typical webpage data) and semi-structured (for e.g. categorized metadata) domains and has had rapidly growing influence on structured data (for e.g. typical business transactions) domains. The XML recommendation is one of the most influential data representation format since the popular relational data model. An example of an XML document ("books.xml") is given in Fig. 3.5:

```
<?xml version="1.0" encoding="UTF-8"?>
<Books>
   <Book>
      <Title>Object-Oriented Conceptual Modeling</Title>
      <Author>
         <LastName>Dillon</LastName>
         <FirstName>Tharam S.</FirstName>
      </Author>
      <Author>
         <LastName>Tan</LastName>
         <FirstName>P. L.</FirstName>
      </Author>
      <Date>1993</Date>
      <ISBN>0-13-712952-1</ISBN>
      <Publisher>Prentice Hall, Australia</Publisher>
   </Book>
   <Book>
      <Title>Data on the Web : from relations to semistructured data and XML</Title>
      <Author>
         <LastName>Abiteboul</LastName>
         <FirstName>Serge</FirstName>
      </Author>
      <Author>
         <LastName>Buneman</LastName>
         <FirstName>Peter</FirstName>
      </Author>
      <Author>
         <LastName>Suciu</LastName>
         <FirstName>Dan</FirstName>
      </Author>
      <Date>1999</Date>
      <ISBN>155860622X</ISBN>
      <Publisher>Morgan Kaufmann, London, UK.</Publisher>
   </Book>
</Books>
```

Figure 3.5: An XML document - books.xml

As stated in the W3C RFC, the design goals for XML include (W3C-XML 2004): (a) it should be easily usable (namely, over the Internet); (b) it should support a wide variety of applications; (c) it should be compatible with its parent language SGML; (d) it should be easier to process XML documents (and easy to write programs that does the processing); (e) it should have zero or absolute minimum optional features; (f) it should be clear and easily understood by humans; (g) it should support quicker design cycle; (h) its design should be formal and concise; (i) it should be easily created; and (j) its markup terseness should be of minimal importance.

As stated earlier, XML uses a restricted version of the SGML which has revolutionized the handling of information in industry, commerce and government organizations. As a variant of the SGML, XML by nature is a meta-grammar and supports: (a) structural and semantic markups; (b) meta-data information; (c) data description; (d) separation of content from presentation; (e) user-defined/custom markups; and (f) flexible link management. As a result, XML is suited well to describing and interchanging of data among heterogeneous databases and systems on the Internet.

In comparison with traditional models (e.g. relational), XML data has considerable flexibility and new challenges for data engineers. The fundamental element to all this is the notion of *data item* vs. XML *tree item*, where, in XML, a tree, the straightforward element is the central aspect. Other characteristics of XML data/document include (Feng, Chang & Dillon 2002; Feng et al. 2003);

- XML has hierarchical tree structures which are more natural, informative and understandable.
- XML data items inherently carry many item specific constraints such as simple/complex content, the notion of ordering and choice and item specific constraints such 'facet".
- XML data items can further be constrained by their context positions, hierarchical level positioning and weak/strong adhesion in the corresponding to other data items.
- In XML data items, relationships among structures and structured values can also be captured and represented.

Note: It should be noted that, from hereon, whenever we refer to XML documents, the reference is to both data-centric and document-centric documents. We do not distinguish between these two in our research.

3.4.1 Document Type Definition (DTD)

An XML document type declaration contains markup declarations which provide the grammar for a class of documents (W3C-XML 2004). The markup declaration is an element type declaration, an attribute-list declaration, an entity declaration, or a notation declaration. The grammar is known as the Document Type Definition (or abbreviated as DTD in short) (W3C-DTD 2001), was one of the first recommendations by W3C for a schema for XML documents. It is part of the XML 1.0 W3C recommendation. The document type declaration can point to an external subset (a special kind of external entity) containing markup declarations, or can contain the markup declarations directly in an internal subset, or it can do both. A DTD for a document consists of both subsets taken together. An example of a simple DTD is given in Fig. 3.6.

```
<?xml version="1.0" encoding="UTF-8"?>
<!ELEMENT Books (Book+)>
<!ELEMENT Book (Title, Author+, Date?, ISBN, Publisher)>
<!ELEMENT Title (#PCDATA)>
<!ELEMENT Author (LastName, FirstName)>
<!ELEMENT Date (#PCDATA)>
<!ELEMENT ISBN (#PCDATA)>
<!ELEMENT Publisher (#PCDATA)>
<!ELEMENT LastName (#PCDATA)>
<!ELEMENT FirstName (#PCDATA)>
```

Figure 3.6: A simple DTD - book.dtd

A DTD is mainly used for describing document-centric XML documents and may be used for some simple applications of data-centric documents. But due to is cryptic structure and limitations, it is not suited well for XML documents that uses mixed content and/or complex content, such as data-oriented constraints and datatyping. Also, because its notation was somewhat different from the XML documents themselves, there was a move to replace it with XML Schema.

3.4.2 XML Schema

XML Schema (W3C-XSD 2001) is arguably one of the most important and complex W3C recommendations for XML. Since its introduction, the exponential growth of XML documents, both in document-centric and data-centric domains, created the need to integrate XML technologies with existing technologies such as data definition tools, database systems, electronic authoring software tool, data exchange (import/export) tools and mediated systems. This triggered a demand to better describe, define and constrain XML document content, stored or exchanged at each individual location, in a manner which is easily understood, encoded and deployed without ambiguity. This demand initiated the process of creating a new schema language for XML; the XML Schema (XSD), a schema definition language for XML, written in XML. An example XSD, the equivalent of the DTD given in Fig. 3.6 is given in Fig. 3.7 (in textual and graphical form), below.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
   elementFormDefault="qualified"
   attributeFormDefault="unqualified">
   <xs:element name="Books">
      <xs:complexType>
         <xs:sequence>
            <xs:element name="Book" type="BookType" maxOccurs="unbounded"/>
         </xs:sequence>
      </xs:complexType>
   </xs:element>
   <xs:complexType name="BookType">
      <xs:sequence>
         <xs:element name="Title" type="xs:string" minOccurs="1"maxOccurs="1" nillable="false"/>
         <xs:element name="Author" type="AuthorType" nillable="false" minOccurs="1"
maxOccurs="unbounded"/>
         <xs:element name="Date" type="xs:string" minOccurs="0"/>
         <xs:element name="ISBN" type="xs:string"/>
         <xs:element name="Publisher" type="xs:string"/>
      </xs:sequence>
   </xs:complexType>
   <xs:complexType name="AuthorType">
      <xs:all>
         <xs:element name="LastName" type="xs:string"/>
         <xs:element name="FirstName" type="xs:string"/>
      </xs:all>
   </xs:complexType>
</xs:schema>
```

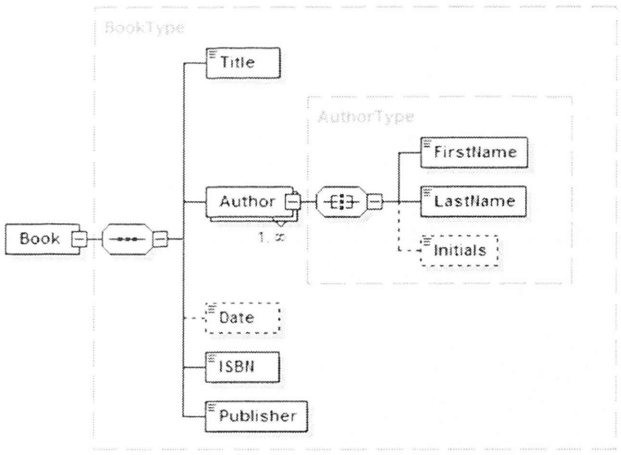


Figure 3.7: A simple XML Schema - book.xsd

The recommendation of XSD is not purely focused on describing and defining XML documents, a task that has already been done by its predecessor DTD, but to overcome many limitations and shortcomings of the DTD and to improve readability of an XML schema definition. DTD is design to describe document-centric XML documents and, by human readability standards, it is cryptic, non-uniform in structure and very difficult to visualize. In comparison to XSD, the other limitations of DTD include: (a) weak data typing; (b) a limited range of content model; (c) problems with validating mixed content type; (d) limited cardinality constraints (e.g. in DTD, only zero, one or many cardinality can be specified); and (e) DTD do not provide a named element or attribute group which is a useful re-usable schema property in XSD.

3.4.3 XML Schema Components

The XSD is composed of thirteen (13) types of schema components that can be broadly grouped into three groups, namely (Feng, Chang & Dillon 2002; Feng, Chang & Dillon 2003): (i) primary components, (ii) secondary components, and (iii) helper components.

The primary components include: (a) simple type definitions, (b) complex type definitions, (c) attribute declarations and (d) element declarations. Here, attribute/element declarations *must* have names, while other type definitions *may* have user defined names.

The secondary components are definitions of: (a) attribute group definitions, (b) identity-constraint definitions, (c) model group definitions, and (e) notation declarations.

The helper components form parts of other components and include five kinds, namely: (a) annotations, (b) model groups, (c) particles, (d) wildcards, and (e) attribute uses.

3.4.4 Other Schemas for XML

There are few other proposals for a schema language for XML. These are individual research or industry sponsored proposals to enable describing and constraining XML better than the document-centric DTD. Some such schema proposals include (Con-W3C 1997; OASIS 2005; W3C-XSD 2001):

- XML-Data (and later XML-Data Reduced or XDR),
- Document Content Description (DCD),
- Schema for Object-Oriented XML (SOX),
- Document Definition Markup Language (DDML),
- Schematron,
- Datatypes for DTDs (DT4DTD),
- Document Structure Description (DSD),
- Regular Language Description for XML (RELAX),
- Document Schema Definition Language (DSDL, ISO/IEC JTC 1/SC34)
- RELAX Next Generation (RELAX NG, part of the DSDL part 2: ISO/IEC 19757-2:2003 standard),
- Tree Regular Expression for XML (TREX),
- Examplotron,
- Hook,
- STEP/EXPRESS (ISO/SC4-10303-0028).

Some of the schemas are still ongoing proposals and some are been replaced by the W3C XSD. For the purpose (and the scope) of this thesis, the notion of XML Schema (or XSD) refers to the W3C XML Schema recommendation (W3C-XSD 2001). Unless specifically stated, a schema for XML refers only to W3C recommendations for a schema definition language for XML.

XML Schema, in addition to providing stronger expressive power than DTD, supports the use of namespaces (W3C-NS 1999), thus providing better modularity.

3.4.5 Schema Document and Instance Document

Table 3.1 shows different terminologies are used to represent some common concepts at varying levels of abstraction for the different paradigms.

Table 3.1: Different Terminologies for Different Paradigms

Level of Abstraction / Paradigm	**Conceptual Level**	**Logical/Schema Level**	**Instance Level**
Relational paradigm	entity	relation	tuple
OO Paradigm	class	object schema	instance object
XML	class	(Schema / DTD) document	XML document or instance XML document

In the relational model (Codd 1970; Date 2003), an entity (and later the static portion of classes, i.e. attributes (Rahayu 2000)) represents a table at the conceptual level, and a relation represents it at the corresponding logical or schema level. A usual norm is to write data instances of a relation (i.e. rows in a table) as *tuples*.

Similarly, as stated earlier, in the OO paradigm (Booch 1993; T.S. Dillon & Tan 1993; Graham, Wills & O'Callaghan 2001), a class represents a collection of instance objects with similar characteristics, such as attributes and methods (T.S. Dillon & Tan 1993). Objects are comparable to tuples.

Since a class captures both static and behavioral aspects, that is, variables that characterize the state of the class are known as *attributes* (static portion), and a set of behaviors associated with the class corresponds to a group of methods (T.S. Dillon &

Tan 1993). Here, attributes describes the class as a whole and the methods alter or determine the states of the class attributes.

At the schema level, object schema serves as a blueprint (or template) for the new instances, i.e. instance objects. It also provides the domain and default-values for the instances. Thus, at present, an instance object is understood to be the instance of a class and the process of creating new instance objects, that is, filling class attributes with attributes using the methods, is called *instantiation*.

However, at the instance level:

- New instance objects are created using the class (and the associated object schema) as a blueprint (or template).
- Instance objects are represented by instance attributes (default or shared).
- Instance objects are represented by the instance methods that are shared by each individual instance objects of the class.

But in the case of semi-structured data (namely XML), as stated earlier, they exhibit both relational and OO characteristics (i.e. in the context of static properties). But, it is generally accepted that, a class (i.e. static portion) may be used to represents an XML document at the conceptual level (Feng, Chang & Dillon 2003). But, at the logical level, as a norm, XML Schema can represent a collection of such classes (and their relationships) that describe and constrain an XML document. Also, since a class that represents an XML document at the conceptual level may not have methods specified (i.e. behavioral portion), the notion of instantiation is not applicable or done externally (e.g. a validating parser) and declaratively such as in (Benjelloun 2004).

Therefore, to date, an XML document *MAY* refers to a schema to check for its validity and it is *NOT* compulsory. The XML document validator/parser may choose to ignore the schema declaration and may only check for well-formed document structure. Thus, at the schema level, an XML document may be optionally represented with some type of *document type definition*, such XML Schema or DTD.

Thus, an instantiation of a (XML) class may be considered as the process of validating (i.e. against a schema) and checking for syntax correctness.

This led to some ambiguity and usage of the term "*document*". Thus, under strictly observed OO paradigm, the term *document*, in the context of XML, corresponds to an instance of an (XML) *class*. But the literal meaning of document is widely used and adapted as a *defacto* standard to represent XML Schema (or DTD) documents also, as:

- they all correspond to textual representations that represent similarities and usage to a computer word-processing document (in contract to DBMS managed tuples and objects), thus the ambiguous usage of the word "documents" to describe both schema and document instances;
- they bear a strong resemblance as an XML Schema is an XML document and the validation and access techniques are similar for both XML and XML schema documents.

Therefore, for the purpose of this thesis, we clarify the ambiguity that exists in using the term "document" in the context of XML and XML Schema.

"*The purpose of a schema is to define a class of XML documents, and so the term "instance document" is often used to describe an XML document that conforms to a particular schema.* " (W3C-XSD 2001)

Thus, in this thesis, unless explicitly stated, the terminologies given in Table 3.2 are used to refer to the corresponding abstract level concepts, to avoid ambiguity in their usage.

Remarks:

- A document type refers to the schema (or logical) level representation that describes and constrains the instance documents.

- An instance document refers to an instance of an (XML) class that conforms to the corresponding document type.
- XML Schema is a collection of document types, constraints and their semantic relationships described using W3C XML Schema.
- A valid XML document is a document that conforms to an XML Schema and is well-formed according to the W3C recommendation given in (W3C-XML 2004).

Table 3.2: Different Terminologies for Different Paradigms

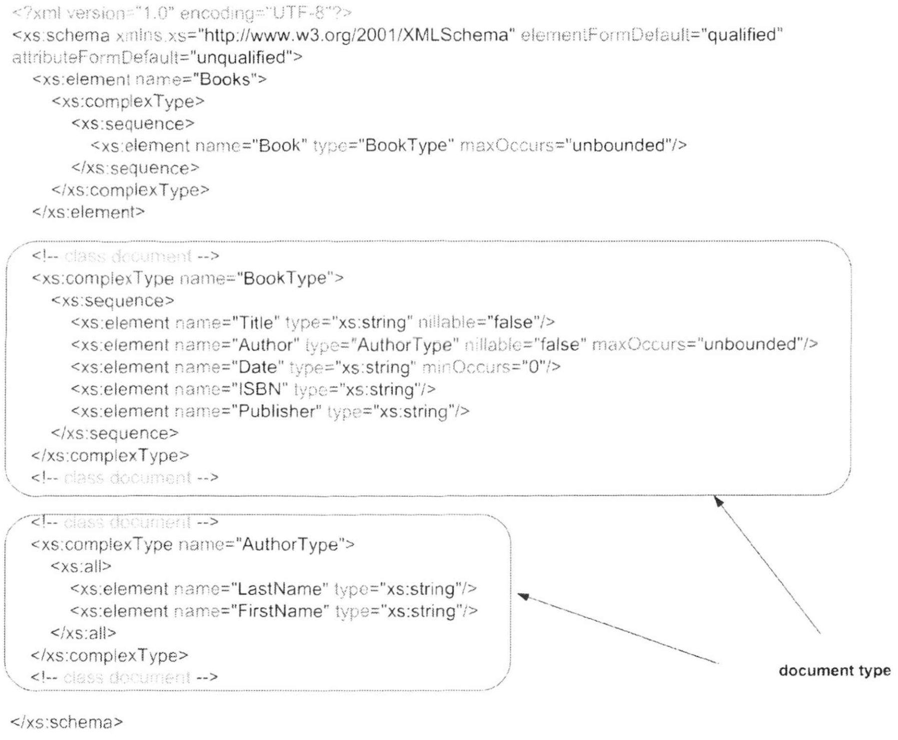
Level of Abstraction / Paradigm	**Conceptual Level**	**Logical/Schema Level**	**Instance Level**
Relational paradigm	entity	relation	tuple
OO Paradigm	class	object schema	instance object
XML	class	***document type***	***instance document***

In practice, semantically related or linked class documents (corresponding to one or more conceptual level classes) are usually described in one XML Schema as abstract types (Fig. 3.8) and additional relationships semantics (e.g. cardinality, keyref, key, etc.) are described as part of the schema as well. Also, semantically related instance documents may be described in one (or more) XML document.

It should be noted that, from hereon, in this thesis:

- The W3C recommendation of a schema for XML is written using initial capital letters as "XML Schema" document (or abbreviated as XSD).

- The concept of schema level representation of a (XML) class is referred to as a "document type". A class document type may be described using any schema language, including XML Schema or DTD.
- An instance document refers to an instance of an (XML) class that conforms to the corresponding *document type.*
- An *XML document* or an *instance XML document* refers to the W3C XML document stated in the (W3C-XML 2004) recommendation.
- Also, unless explicitly stated, a (XML) Schema instance refers to an XML document.



```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified"
attributeFormDefault="unqualified">
  <xs:element name="Books">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="Book" type="BookType" maxOccurs="unbounded"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>

  <!-- class document -->
  <xs:complexType name="BookType">
    <xs:sequence>
      <xs:element name="Title" type="xs:string" nillable="false"/>
      <xs:element name="Author" type="AuthorType" nillable="false" maxOccurs="unbounded"/>
      <xs:element name="Date" type="xs:string" minOccurs="0"/>
      <xs:element name="ISBN" type="xs:string"/>
      <xs:element name="Publisher" type="xs:string"/>
    </xs:sequence>
  </xs:complexType>
  <!-- class document -->

  <!-- class document -->
  <xs:complexType name="AuthorType">
    <xs:all>
      <xs:element name="LastName" type="xs:string"/>
      <xs:element name="FirstName" type="xs:string"/>
    </xs:all>
  </xs:complexType>
  <!-- class document -->

</xs:schema>
```

Figure 3.8: A W3C XML Schema example

3.4.6 Namespaces

XML relies on a mechanism called namespaces, to unambiguously distinguish different markup vocabularies (i.e. elements and attributes) in an XML document, as one single XML document may be defined and used by more than one user application.

Definition 5.15. An XML namespace is a collection of names, identified by a URI reference (RFC2396), which are used in XML documents as element types and attribute names (W3C-NS 1999).

3.5 Documents, Schema, Operators & Query Expressions.

A typical semi-structured data model is usually represented as a rooted directed graph (Feng, Chang & Dillon 2002). In the case of an XML document, it consists of a set of elements connected in a user-defined hierarchy and has user-defined names.

Since an XML document structure may be represented as a tree which is usually an acyclic graph (Feng, Chang & Dillon 2002), we represent XML document model (XDocM) using a graph model with a set of *vertices* (that corresponds to tags), a set of *edges* and a special vertex called *root* node.

A vertex in XDocM is a representation of the elements and values. It can be of complex type denoting user defined content or of simple type representing atomic data types such as integer, char, etc.

A vertex v_{ertex} is a pair with a unique id v_{id} that uniquely identifies a vertex in the graph hierarchy and its body or content $v_{content}$. The body content is of type either simple content ($s_{content}$) that holds atomic values or complex content ($c_{content}$) that hold user defined types or constructional content such as set, list etc.

$$v_{ertex} = (v_{id}, v_{content}) \qquad \textbf{(3.24)}$$

3.5.1 Root Node

Definition 3.16. A root node is the topmost node (i.e. minimal vertex) in an XML document connecting to zero or more nodes via a series of edges.

$$v_{root} \in v_{ertex} \qquad (3.25)$$

thus,

$$v_{root} = (v_{id}, v_{content}) \qquad (3.26)$$

Also, since root node is the minimal vertex in the XDocM,

$$v_{root} = Min_{i=0}^{s}(v_{ertex}^{s}) \qquad (3.27)$$

that is, $v_{id} = 0$.

3.5.2 Edges

Definition 3.17. An edge is an *ordered* pair with two *adjacent* vertices.

In our work, it is assumed that all edges are *well-formed*. Let e_{dge} denote an edge of two adjacent vertices (i.e. source, target vertices) v_s, v_t with vertex ids v_{id}^{s}, v_{id}^{t} respectively. Therefore an edge may be written as:

$$e_{dge} = (v_s, v_t) \qquad (3.28)$$

where

$$v_s, v_t \in v_{ertex} \qquad (3.29)$$

and

$$(v_{id}^{s} < v_{id}^{t}) \wedge (v_{id}^{s} \neq v_{id}^{t}) \qquad (3.30)$$

Also, it is commonly assumed (in graph theory) that:

$$v_{ertex} \wedge e_{dge} = \phi \qquad (3.31)$$

There is also a common form of writing an edge, as:

$$(v_s(e), v_t(e)) \qquad (3.32)$$

where,

$$e \in e_{dge},\ v_s(e), v_t(e) \in v_{ertex} \qquad (3.33)$$

3.5.3 An XML Document

Definition 3.18. An XML document is a triple representing a rooted tree, with one root node, set of interconnected vertices and set of edges connecting the vertices.

Let ψ_{XML} denote an XML document. Also let v_{root} be the root node, V be a set of connected vertices and E be a set of edges, such that;

$$V = \{v_1, v_2, \ldots v_k, \ldots v_n\} \qquad (3.34)$$

$$\forall v_k \in V : v_k = (y) \quad with \quad y \in v_{ertex} \qquad (3.35)$$

$$E = \{e_1, e_2, \ldots e_i, \ldots e_{n-1}\} \qquad (3.36)$$

$$\forall e_i \in E : e_i = (y) \quad with \quad y \in e_{dge} \qquad (3.37)$$

and

$$\forall v_{root} \in v_{ertex} \qquad (3.38)$$

Therefore an XML document (model) may be written as:

$$\psi_{XML} = (v_{root}, V, E) \tag{3.39}$$

It should be noted that, a more elaborate definitions and notations are possible for writing an XML document structure. Here, we use a simple notation to represent a generic XML document tree which may be extended depending on one's need.

In simple terms, an XML document may be described as a linearization of a tree structure (Con-W3C 1997; W3C-XML 2004). At every node in the tree there are several character strings. The tree structure and the character strings together form the information content of an XML document.

3.5.4 An XML Schema Document

As we have stated that an XML Schema is itself an XML document. Thus, an XML Schema document structure can be represented as a (XML) tree. Thus using equation 3.39, we can write an XML Schema as:

$$\psi_{XSD} = (v_{root}, V, E) \tag{3.40}$$

Let $\chi_{dataType}$ be a set of data types that is defined in a given XML Schema χ_{xsd}. Also, since an XML Schema is a description and definition of a collection of real-world data types and their relationships of various namespaces N_{spaces}, let χ_{xsd} be an XML Schema, $\chi_{simpleType}$ be a set of XSD build-in atomic data types (e.g. integer, string, date, long, etc) used in χ_{xsd}, and $\chi_{complexType}$ be a set of complex or user-named data types (e.g. in Fig. 3.7, Book, Author, etc.), defined in χ_{xsd}. $\chi_{simpleType}$ and $\chi_{complexType}$ corresponds to XML simple ($s_{content}$) and complex ($c_{content}$) content respectively.

Definition 5.19. An XML Schema χ_{xsd} is a triple χ_{name}, $\chi_{dataType}$ and $\chi_{constraint}$, where χ_{name} is the name of the XML Schema χ_{xsd} (i.e. the name of the resulting XML schema file, a valid W3C XML document name), $\chi_{dataType}$ is a set of simple ($\chi_{simpleType}$) and complex ($\chi_{complexType}$) type definitions for XML elements/attributes, and $\chi_{constraint}$ is a set of constraints defined upon XML elements/attributes. Both $\chi_{dataType}$ and $\chi_{constraint}$ are expressed in XML Schema language.

$$\chi_{xsd} = (\chi_{name}, \chi_{dataType}, \chi_{constraint}) \qquad \textbf{(3.41)}$$

3.5.5 Path

Here, we look at the concepts of path, path length and original path in an XML document.

Let p_{ath} denote a path with starting v_1 vertex and ending vertex v_n of a sequence of connected vertices $v_1, v_2, \ldots. v_i, \ldots. v_n$.

Definition 3.20. The path p_{ath} is a *sequence* of vertices with a *start* vertex, an *end* vertex, and for any adjacent vertices, there exists a well-formed edge.

$$p_{ath} \mapsto (v_1, v_2, \ldots. v_i, \ldots. v_n) \qquad \textbf{(3.42)}$$

where

$$v_i \in v_{ertex} \text{ and } 1 \leq i \leq n \qquad (3.43)$$

For any two *consecutive* vertices v_i and v_{i+1} in the path p_{ath} ,

$$(v_i, v_{i+1}) = e \text{, where } e \in e_{dge} \qquad (3.44)$$

Also, using equation 3.32, it can be shown that, given a sequence of edges $(e_1, e_2, \ldots\ldots e_i, \ldots\ldots e_n)$, a path p_{ath} may be written as (Abiteboul, Buneman & Suciu 1999):

$$v_t(e_i) = v_s(e_{i+1}) \text{, where } 1 \leq i \leq n \qquad (3.45)$$

where

$$e_i \in e_{dge} \text{, } v_s(e_i), v_t(e_i) \in v_{ertex} \qquad (3.46)$$

Conversely, a path $p_{ath} \mapsto$ can be also stated as, the path p_{ath} that it goes through vertex v_i , where $1 \leq i \leq n$. Let denote the length of a path as $length(p_{ath})$.

Definition 3.21. The length of a path $length(p_{ath})$ is defined as the number of vertices that a path goes, which is equal to the total number of vertices between the start and end vertex inclusive.

$$length(p_{ath}) = \sum_{i=1}^{n} v_i = n \qquad (3.47)$$

Definition 3.22. A path p_{ath}^{o} is said to be an *original* path, if and only if, the start vertex v_1 has no incoming edges.

In simple terms, an original path usually has the root vertex as its starting vertex. Thus, we can show a root as:

Definition 3.23. The root is the *start* vertex of an original path p_{ath}^{o} such that the path length is 1 (i.e. $length(p_{ath}^{o}) = 1$).

Let vertex v be the *end* vertex to a collection of original paths $p_1^o, p_2^o, \ldots\ldots, p_i^o, \ldots p_s^o$.

Definition 3.24. The depth of a vertex v is the maximal length of the original path in $\{p_1^o, p_2^o, \ldots\ldots, p_i^o, \ldots p_s^o\}$ (Feng, Chang & Dillon 2002).

$$depth(v) = Max_{i=1}^{s}(p_i^o) \qquad (3.48)$$

The depth of the root, which is a vertex without any incoming edges, is always assumed to be 1. Thus, we can also define the root as:

Definition 3.25. The root is a vertex with vertex depth of 1 (i.e. $depth(v_{root}) = 1$).

3.5.6 Operators

Operators are basic procedures that operate on one or more (usually *ordered*) data sources or operands to provide alternate data composition or data extraction.

Let λ denote a generic operator and exp^{λ} denote an expression. An expression is an optional predicate statement made to restrict or constrain an operator's results, where $exp^{\lambda} \in C_{constarint}$. Also, let O_p denote an operand. Thus, the general form of an operator is show as:

$$\lambda(O_p^1, [O_p^2, ..., O_p^k, ..., O_p^n])_{([exp^{\lambda}])} \tag{3.49}$$

In the case of unary operators, the minimum number of operand permitted is one, i.e. $k \geq 1$. In the case of binary operators, it is always $k \geq 2$. Also,

$$\forall exp \in exp^{\lambda} : exp = (\rho) \quad with \quad \rho \in C_{constraint} \tag{3.50}$$

3.5.7 Query Expression

A query expression is an *ordered* sequence of operators with optional restrictions or constraints, expressed in the form of a sequence of expressions exp^Q, applied to the query result set ($exp^Q \in C_{constarint}$).

$$Q_{uery} = (\lambda_1, \lambda_2, ..., \lambda_k, ...\lambda_n)_{([exp_1^Q, exp_2^Q, ...exp_a^Q....exp_m^Q])} \tag{3.51}$$

where,

$$\forall \lambda_j^i \in \lambda_k : \lambda_j^i = (\rho) \quad with \quad \rho \in \lambda \tag{3.52}$$

and

$$\forall exp \in exp_a^Q : exp = (\rho) \quad with \quad \rho \in C_{constraint} \tag{3.53}$$

3.6 Views

In data engineering, the notion of views is essential in databases, as it allows various users to see data from different viewpoints. This is made possible by data "abstraction" and data "partitioning" provided by the view model in a language specific, well-defined data model. Since the introduction of the successful view model for the relational data model (Codd 1970, 1990; Date 2003; Elmasri & Navathe 2004), motivation for views has changed over the last two decades. As stated in Chapter 1, it is apparent that the uses and applications of views are realized more than originally intended, as proposed by Codd (i.e., the 2-Es - data Extraction and Elaboration). However, despite the usefulness of views and their widespread applications, the notion of view concept remains as a data language and model dependent, low-level (instance) construct. That is, as described in Chapter 1 (and elaborated more in Chapter 2), a view is a *virtual* table (or class in OO) in the structured data domain. Conversely, in conceptual modelling paradigms such as OO or E-R/DFD, the modelling notation and/or languages provide no semantics or constructs to capture view formalisms at the conceptual (or logical) level.

As we have stated in Chapters 1 and 2, in database literature, it is widely accepted that a (traditional) view is a *named* query expression. Let V_{iew} denote a view with V_{name} denoting its name and Q_{uery}^{V} denote a query expression. Therefore, we can denote a (traditional) view as:

$$V_{iew} = (V_{name}, Q^{V}_{uery}) \qquad (3.54)$$

where,

$$\forall q_{uery} \in Q^{V}_{uery} : q_{uery} = (q) \quad with \quad q \in Q_{uery} \qquad (3.55)$$

3.7 Views for XML

As we have clearly stated in Chapter 1, the notion of view is an important concept in data engineering and may be utilized in many applications, from user access control to dimensional modeling. Thus, in summary, look views in XML are useful as a mechanism, for:

- extracting of sub-tree/(s) from one or more XML documents (Abiteboul, Goldman et al. 1997; Aguilera et al. 2002);
- extracting of (sub) document structure without loss of original document semantics and structure;
- providing grouping of semantically (not synthetically) related document sub-structures that is of meaning to users (and user applications);
- providing access control for document and document structures at different levels of document hierarchy (Steele et al. 2005a, 2005b);
- providing virtual constructs (e.g. data cubes, dimensions, application programming interfaces, etc.) or wrappers for stored XML documents;
- providing scalability to applications that handle large documents and document structures, such as data warehousing and data mining(Aguilera et al. 2002; Fankhauser & Klement 2003; Feng & Dillon 2005; Feng et al. 2003; Tan et al. 2005);

- providing structure (or super structures) and fine-grained data access to existing distributed heterogeneous documents, such as web, relational, XML etc. (i.e. mixed content);
- integrating heterogeneous data sources (structured, semi-structured and incomplete) (Abiteboul, Goldman et al. 1997; Aguilera et al. 2002);
- providing distributed meta-data for applications such as middleware (Abiteboul et al. 2002; MIX 1999), data warehouses (Jeong & Hsu 2001; Lucie-Xyleme 2001; Medina, Luján-Mora & Trujillo 2002) etc.;
- extracting structures and storage representations from large Semantic Web applications such as an ontology and sub-ontology base (Bhatt et al. 2004; Wouters 2006; Wouters et al. 2004);
- providing document markup for structured data for data dissemination, distribution and web publication (Lucie-Xyleme 2001);
- providing unified document structure and access for heterogeneous XML documents that are stored using different storage models (e.g. relational, OO, file system etc.);
- providing query operations such as joins, aggregates, summation on original or sub XML documents.

Thus, given these useful properties of XML views, one should look at how such a useful view mechanism should be proposed, in the context of XML. As we have shown in equation 3.54, most existing view mechanisms (including that which exists for XML views), are considered as a named query. This is mainly due to the popularity of relational views and the implicit view mechanism that was derived or extended (e.g. OO views (Abiteboul & Bonner 1991; Kim & Kelly 1995), XML views (Abiteboul 1999; Abiteboul, Goldman et al. 1997), O-R views etc.) from the relational views (Fig. 3.8). As we have discussed in Chapter 2, such view mechanisms lack many crucial aspects of an XML view mechanism that is capable of providing all the perceived benefits outlined above.

This is mainly because of the semi-structured nature of XML and the characteristics that are associated with it. The shortcoming of the existing view mechanism adapted (or extended) for XML and other XML views, may be grouped and studied under three main topics, namely:

(i) XML document structure and data model

(ii) Query languages

(iii) Application and integration into existing and emerging data paradigms

The following sections discuss these topics in detail.

3.7.1 XML Document Structure and XML Views

XML document structure is semi-structured in nature and exhibits document-centric characteristics. Conversely, a structured data item, e.g. a person's *first name,* conforms to one system wide data type (or schema type) that defines its data format, domain and possible values. These structured data items are usually singular (or simple) or single valued. Also, such data values can be accessed via query language, which first determine the predefined datatype (or schema) from the global schema and execute the query issued against the data sets (or tuples). But advancements in OO and database technologies have enabled one to have complex data items (or Abstract Data Types (ADT)) and provide techniques to access such complex data structures to be accessed via query languages (T.S. Dillon & Tan 1993). The popularity and the efficiency of the relational model have made the structured data as the de facto standard for many Information Systems. The success of the relational model is evident from two facts: (i) though successful in modeling and programming, unpopularity of native OO databases and (ii) successful adaptation of OO (logical) data models using core relational database engines.

However, in contrast to structured data, an XML document needs to be valid and well-formed, i.e., its data description and constraining tags must be properly nested and conform to a particular Schema/DTD). Due to this nature, many researchers consider XML to be a variant of semi-structured data model, where document data structure may vary from document to document (e.g. repeated or missing sequence) for a given user-defined schema or structure (Abiteboul 1999;

Abiteboul, Buneman & Suciu 1999; Feng, Chang & Dillon 2002; Feng, Chang & Dillon 2003).

Management of semi-structured data by highly-structured modeling techniques, such as relational and object-oriented models, not only results in a very complicated logical schema, but also requires much effort and frequent schema modifications. Thus, XML is said to exhibit the following characteristics that are different from traditional structured data, due to the inherent flexibilities of XML in both structure and semantics (Feng & Dillon 2005; Feng et al. 2003). They are:

- XML data has a more complex hierarchical structure than a structured database record
- XML document elements have contextual positions which thus carry the order notion
- XML documents trend to be much bigger than traditional data records due to embedded data descriptions
- Though capable, XML document structures usually do not reflect real-world semantic relationships and mostly represent synthetic relationship structures.

Therefore, existing view mechanisms including most XML views (and other extended mechanisms) are not well-suited for handling such a complex document structure in a meaningful manner as, they are supposed to handle XML data in an XML-enabled database environment.

3.7.2 Data Model, Query and XML Views

In comparison to structured data, there is no agreed standard on the native XML storage model yet. Thus, the XML view mechanisms that exist usually use some form of structured data models (e.g. relational, OO, etc.) (Abiteboul 1999; Abiteboul, Goldman et al. 1997; Abiteboul, Quass, McHugh, Widom et al. 1997; YB Chen, TW Ling & ML Lee 2002) proposed has to be generic enough to support emerging new XML storage standards.

In regards to view query language, in the relational model (Fig. 3.9), there is a well-defined, industry standard query language; the Structured Query Language (SQL-92) for data definition, manipulation and access, including support for view specification and definition. But in the case of XML, only recently, XQuery (W3C-XQuery 2004) specification was published and the successful implementation and industrial adoption is still in their initial stages. Unlike SQL, XQuery is not developed or flexible enough to support view construction yet. Therefore many researches have extend XQuery (YB Chen, TW Ling & M-L Lee 2002) or use their own customized query language (like LOREL(Abiteboul, Quass, McHugh, Widom et al. 1997)) to define XML views at the data language level. But so far, none of them are generic enough to implement in a language independent XML views that are capable of adopting emerging new query language standards.

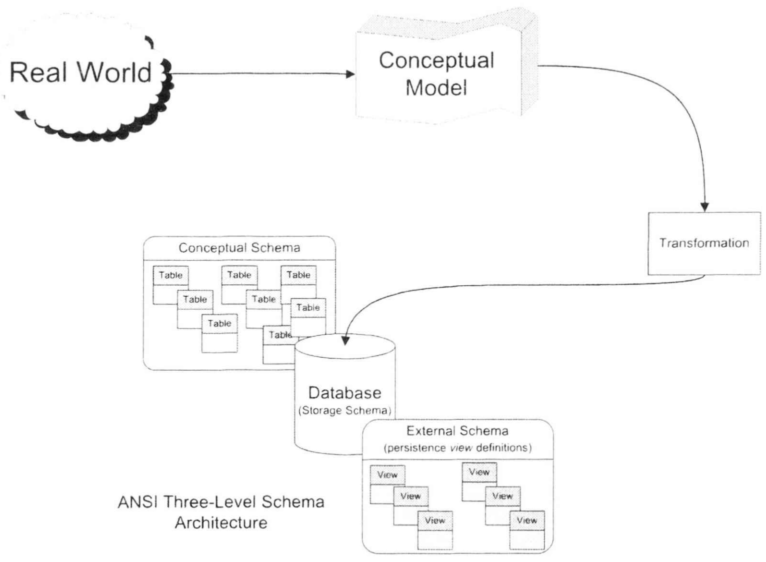


Figure 3.9: Views in traditional data engineering approach

Another issue associated with XML view specific querying is the nature of query language itself. In the relational model, ad-hoc querying is one of the key features behind the SQL standards. In XML, there can be iteration within hierarchical structures. Therefore, there exist few problems for defining and querying XML (and view) documents using traditional ad-hoc like query languages (or extensions of such languages). This is mainly due to query language limitations and the strong dependency of views on such languages. Some such problems, in the context of XML, are given below.

(i) Dual property: in XML, for a given (view) query expression, it should be able to construct result sets from stored both data-centric and document-centric XML documents without modifications.

(ii) Heterogeneity: in contrast to relational data, where all tuples in relations are of the same structure, XML provides varying and (non-atomic) complex structures. Therefore, in a stored XML document, depending upon such structures and embedded constraints (reference, choice, all, sequence etc), sub-elements of XML hierarchy may be missing and/or repeated. At present, many existing XML *view definition languages* need complex data definition and specification formalisms to perform such tasks.

(iii) Uniqueness: each tuple in a relation can be uniquely accessed due to key-base constraints (primary key, foreign key) defined for them. In the case of XML, its semi-structured nature prevents similar constraints from being placed. But in comparisons with relational models, defining and validating such a key mechanism for XML is a computationally expensive and complex task, as unlike relational, the XML structure is hierarchical. Also, unlike relational, XML is not dependent on keys and normalization but on node hierarchy and ordering. But there exists some work on indexing, "normalization" for XML and defining key-based constraints to improve query processing for XML (Blanken 2003).

(iv) Ambiguity: an XML document, due to its self-descriptiveness, can be vague in structural definition. Therefore, one document may have multiple schemas defined. Therefore, querying and defining views for such documents is a difficult task, and thus view definition may vary from one language to another.

(v) Ordering: data extraction and elaboration in XML (and in many semi-structured data) documents should keep the original order of structures. Thus, query language (and views) should be aware of such constructs and process them accordingly. Again, at present, in the context of view definition and querying, due to non-defined standards, tasks such as specifying and processing ordering vary from one view definition language to another.

(vi) Logical or schema constraints: in many XML documents, constraints such as element repetition, ordering etc. are only visible and constrained at the schema level than being embedded within the document elements. Thus, in such circumstances, the view definition (and the query) language should be able to access and process such logical level constructs to keep the XML document semantics intact.

(vii) Customization: most existing view definition languages provide view customization only at the instance (or result) level. They do not allow customization of XML document structures at the higher level (e.g. schema) that may be required for processing some queries and/or document constraints (e.g. model integration, abstract query processing, etc.).

(viii) Language independence: most existing XML view mechanisms do not support view definition that is independent of view specification languages, thereby lacking the flexibility required in extracting hierarchical data structures without loss of syntactical XML document semantics that is required for XML (and in general for semi-structured data)

Thus, in summary, it is clear from the aforementioned that a view mechanism for XML may be benefit more if:

- it is independent of the view specification language;
- it provides view specification (and possibly definition) at a higher level (e.g. schema) than at the instance level;
- the view specification and definition are of native (XML) in nature;
- it adheres to generic definition that is deployable in a varity of present and emerging XML technologies and storage models, as XML (and semi-structured data) standards are an on going process;
- it supports XML data extraction and (preferable) elaboration at varying levels of abstraction, such as instance, query and schema levels.

3.7.3 Applications and Integration of XML Views

Integration and adaptation of XML views is another key issue, as it is one of the most important aspects of views, where, due to their flexibility, views may be incorporated into any database application without major changes to the application interface, database schema, data and the storage structure.

In relational databases, views are part of the external schema (Fig. 3.9), where persistent view definition is instantiated when a query is issued against the view. Conversely, new view definitions can be created and/or deleted without affecting the consistency, structure and storage space of the overall database schemas and data. This characteristic of views helps to design data intensive architectural constructors such as data cubes (Humphris, Hawkins & Dy 1999; Ponniah 2001), operational data stores (Inmon, Imhoff & Battas 1996; Kimball & Ross 2002), persistent data interfaces etc. Thus, to apply such concepts to XML is a challenging and difficult task as XML is independent of one centralised (or unified) data schema, as the XML data structure and schema are dynamic in nature.

3.7.4 Properties of an Ideal View Mechanism for XML

In the sections above, we highlighted some of the benefits and issues associated with views for XML. Based on such discussion, here we outline some of the envisaged properties of an *ideal* view model for XML that is being considered in this research. They are:

- A view mechanism for XML should be independent of a fixed data model, and rely on dynamic XML Schema/DTD like mechanism for data description and elaboration. This is a contradistinction to some of early work on XML views, such as (Abiteboul 1999), where authors argue that an XML view model should depend on ODMG (Cattell et al. 2000) like structured data model. The argument for this is that, XML is well adopted and accepted due to its data model independence and the rich data description (semantics) provided by its self-descriptive tags. Adapting an ODMG like data model for XML and XML views may lose its purpose and introduce complex schema (or logical wrapper) processing and management, resulting in loss of data semantics, meaning and performance in various applications.

- A view mechanism for XML should be independent of one particular query language (e.g. SQL) or propriety data languages, and should be able to support multiple query languages as, one core purpose of XML is interoperability, where one source XML data segment/document may need to be instantiated at multiple target IS sites, by multiple query/data languages, generating similar data sets. Also, to accommodate fast emerging, robust and efficient query and data (or domain) languages (e.g. LOREL (Abiteboul, Quass, McHugh, Widom et al. 1997), RDQL (W3C-RDQL 2004), SPARQL(SPARQL 2004), SQL 03(ANSI/ISO 2003) etc.), the view model (and the view instantiation) should be independent of one specific query/data language.
- As we described in Chapter 1, an XML view may be a base for defining other views (view of a view) and may have to stratify queries issued against it. Thus, a view mechanism for XML should provide validation (YB Chen, TW Ling & ML Lee 2002) in addition to well-formed view instances. That is, a XML view instance should be a "self-contained" or "mini" XML document with its own validation mechanism (such as view DTD or Schema), as unlike in structured data views, there exists no global schema to constraint and validate XML view instances.
- The XML view instance, should not only provide individual data elements, but also the sub-tree hierarchy (without loss of data semantics and integrity) from the original stored document/(s) structure.
- XML data items inherently carry many item specific constraints such as simple/complex content, the notion of ordering and choice and item specific constraints such as "facet".
- A view mechanism for XML should support easy integration into existing and new XML applications and support construction of architectural constructs such as data cubes, dimensional data models and XML (data) interfaces.

In the next section, we clearly define the problem of providing XML views with the above perceived properties.

3.8 The Problem Definition

The problem solved in this thesis consists of five related sub-problems and these are;

(1) Develop an approach that permits separation of *implementation* aspects and *conceptual* aspects of views so that there is clear separation of concerns thereby allowing analysis and design of views to be separated from their implementation.

(2) Define representations to express these views in a conceptual model.

(3) Define conceptual operators to permit the encoding of construction of views at a higher level of abstraction than being tied to a specific data manipulation language.

(4) To define a view development methodology for XML views that utilizes the conceptual models and carries out automated transformation to an XML schema for the views as well as a representation of the view construction in an appropriate XML query language such as XQuery (W3C-XQuery 2004).

(5) To validate and apply the concepts, methods and transformations developed in (1) to (4) above to specific domains, particularly:

a. Designing a view-driven XML document warehouse

b. A view mechanism for Semantic Web and

c. Views-driven website and web portal engineering

In summary, each problem described above (1- 5) will satisfy one or more of the view properties described in section 3.7.4 above, and are elaborated upon in the context of our research below.

(i) The view specification will be based on real-world objects, relationships and constraints and not just on stored, instance XML documents. That is, a view will be considered as a *first-class citizen* in the conceptual model.

As we have stated so far, at present, existing view models, their definition and specification languages, such as (a) SQL (Date 2003), (b) OQL dialects such as (Abiteboul & Bonner 1991; Cattell et al. 2000; Kim & Kelly 1995), (c) LOREL (Abiteboul, Quass, McHugh & Widom 1997; Abiteboul, Quass, McHugh, Widom et al. 1997), (d) Object Constraint Language (OCL) (OMG-OCL 2003) based view specification models such as (Balsters 2003) and (e) Other declarative view specification languages such as (Braga, Campi & Ceri 2005; YB Chen, TW Ling & ML Lee 2002) do not provide specification mechanism to have views as first-class citizens.

(ii) The view specification and definition will be independent of programming and/or query language constructs and syntax.

(iii) Conceptual and schemata semantics will be part of the view definition.

(iv) The view model for XML will incorporate a comprehensive design methodology to precisely model, design and instantiate XML views.

(v) The XML view data generated will be valid and well-formed.

(vi) The XML view model will provide an explicit mechanism to describe and constraint (elaboration) view results.

(vii) The XML view will be capable of producing results for queries issued against its definition

(viii) The XML view definition will support construction of other views (i.e. view of a view) based on its own definition.

(ix) The XML view definition will support *ordering* and *path distinctions* based on the original stored XML document structure/(s).

(x) The XML view definition will be easy to integrate and adaptable to the current and emerging XML standards (e.g. XQuery(W3C-XQuery 2004), SQLX(ANSI/ISO 2003), OWL (W3C-OWL 2004), RDQL(W3C-RDQL 2004), etc.).

(xi) The XML view definitions will be capable of constructing architectural constructs for data domains such as data warehouses, without complex procedural programming.

3.9 Evaluation of Feasible Solutions

To solve the problem views for XML, there are a number of feasible solutions considered in this thesis. They are:

(i) XML-enabled solutions

a. relational view mechanism

b. Object-Relational (O-R) view mechanism

c. Object-Oriented (OO) view mechanisms for XML

(ii) Declarative or notational language based view definitions

(iii) Graph structured models with support for XML (tree) structures

(iv) An OO based generic XML view mechanism conceptual and logical extensions.

Relational models generally are not expressive enough to provide the required semantics to model the complex semantics of XML and semi-structured data. Whilst OO models have much more expressive semantics that have been effectively used for modeling domain semantics, enhancements of them are proposed in (Feng, Chang & Dillon 2003; Xiaou et al. 2001a; Xiaou et al. 2001b) to allow one to model semi-structured data and XML documents. However, no attempts to date have been proposed to use these for view definitions in the way suggested above in section 3.8. We will, however, examine extending this as a basis for view definitions.

Graph models on their own have no appropriate semantics but are useful in developing the connections or links between semi-structured data. An XSemantic net model which has the required semantics, has been proposed in (Feng, Chang & Dillon 2002), and has been shown to effectively model XML document data. We explore this also as the basis for view definitions.

3.10 Conclusion

In this chapter we formulated the problem that is being addressed in this thesis. First, some basic concepts, definitions and their formal semantics were given, followed by the shortcomings of traditional views and the problems that exist in adapting such a view mechanism for XML. After clearly establishing the need for an expressive view mechanism, in the context of XML, an ideal view mechanism and its properties were given.

This ideal XML view mechanism and its properties lead to the formulation of the problem being addressed in this research. This problem formulation is divided into four major aspects, namely: (a) the separation of conceptual and implementation semantics in a view mechanism; (b) a view mechanism with formal semantics for XML, with conceptual and logical extensions; (c) a detailed design and transformation methodology for modelling views for XML; and (d) validation of the proposed XML views in real-world scenarios.

Finally, we evaluated some of the existing feasible solutions that are capable of solving one or more of the problems described above. After careful consideration, one solution was selected as the overall solution, which will be addressed in the following chapters.

The next chapter will provide an overview of the solution proposed in this research, followed by chapters that provide detailed discussion of one or more sub-components of the solution.

References

Abiteboul, S 1999, 'On Views and XML', *Proceedings of the eighteenth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems (PODS '99)*, ACM Press New York, NY, USA, Philadelphia, Pennsylvania, USA, pp. 1-9.

Abiteboul, S, Benjelloun, O, Manolescu, I, Milo, T & Weber, R 2002, 'Active XML: A Data-Centric Perspective on Web Services', *BDA*.

Abiteboul, S & Bonner, A 1991, 'Objects and Views', *ACM SIGMOD Record, Proceedings of the International Conference on Management of Data (ACM SIGMOD '91)*, vol. 20 (2), ACM Press New York, NY, USA.

Abiteboul, S, Buneman, P & Suciu, D 1999, *Data on the Web : from relations to semistructured data and XML*, Morgan Kaufmann, London, UK.

Abiteboul, S, Goldman, R, McHugh, J, Vassalos, V & Zhuge, Y 1997, 'Views for Semistructured Data', *Workshop on Management of Semistructured Data*, USA.

Abiteboul, S, Quass, J, McHugh, J & Widom, J 1997, 'Lore: A Database Management System for Semistructured Data', *SIGMOD Record*, vol. 26(3), ACM, pp. 54-66.

Abiteboul, S, Quass, J, McHugh, J, Widom, J & Wiener, J 1997, 'The Lorel Query Language for Semistructured Data', *International Journal on Digital Libraries*, vol. 1, no. 1, pp. 68-88, April 1997.

Aguilera, V, Cluet, S, Milo, T, Veltri, P & Vodislav, D 2002, 'Views in a Large-Scale XML Repository', *The VLDB Journal — The International Journal on Very Large Data Bases*, vol. 11(3), no. 3, pp. 238-55

ANSI/ISO 2003, *ANSI - SQL 2003*, ANSI / ISO.

Balsters, H 2003, 'Modelling Database Views with Derived Classes in the UML/OCL-framework', *The Unified Modeling Language: Modeling Languages and Applications (UML '03)*, vol. 2863, Springer, USA, pp. 295-309.

Benjelloun, O 2004, 'Active XML: A data centric perspective on Web services', PhD thesis, Paris XI University, Orsay, France.

Bhatt, M, Flahive, A, Wouters, C, Rahayu, W, Taniar, D & Dillon, TS 2004, 'A Distributed Approach to Sub-Ontology Extraction', *18th International Conference on Advanced Information Networking and Applications (AINA'04) - Volume 1*, vol. 1, IEEE Computer Society, Fukuoka, Japan, pp. 636-41.

Blanken, HM 2003, *Intelligent search on XML data: applications, languages, models, implementations, and benchmarks*, Lecture notes in computer science, 2818., Springer, Berlin ; London.

Booch, G 1993, *Object-oriented analysis and design with applications*, 2nd edn, The Benjamin/Cummings series in object-oriented software engineering., Benjamin/Cummings Pub. Co.; Addison-Wesley, Redwood City, Calif. Reading, Mass.

Braga, D, Campi, A & Ceri, S 2005, 'XQBE (XQuery By Example): A visual interface to the standard XML query language', *ACM Transactions on Database Systems (TODS)*, vol. 30(2), no. 2, pp. 398-443

Cattell, RGG, Barry, DK, Berler, M, Eastman, J, Jordan, D, Russell, C, Schadow, O, Stanienda, T & Velez, F (eds) 2000, *The Object Data Standard: ODMG 3.0*, Morgan Kaufmann.

Chang, E 1996, 'Object Oriented User Interface Design and Usability Evaluation', Doctor of Philosophy (Ph.D) thesis, La Trobe University, Melbourne, Australia.

Chen, YB, Ling, TW & Lee, ML 2002, 'Designing Valid XML Views', *Proceedings of the 21st International Conference on Conceptual Modeling (ER '02)*, vol. 2503, eds S Spaccapietra, ST March & Y Kambayashi, Springer-Verlag London, UK, Tampere, Finland, pp. 463 - 78.

Chen, YB, Ling, TW & Lee, M-L 2002, 'A Case Tool for Designing XML Views', *Second International Workshop on Data Integration over the Web (DiWeb '02)*, ed. Ze Lacroix, University of Toronto Press, Toronto, Canada, pp. 47-57.

Codd, EF 1970, 'A Relational Model of Data for Large Shared Data Banks', *Communications of the ACM*, vol. 13(6), no. 6, pp. 377-87

Codd, EF 1990, *The Relational Model for Database Management: Version 2*, Addison Wesley Publishing Company.

Con-W3C 1997, *World Wide Web Consortium (W3C), http://www.w3.org/*, The World Wide Web Consortium (W3C).

Date, CJ 2003, *An introduction to database systems*, 8th edn, Pearson/Addison Wesley, New York.

Dillon, TS, Chang, E, Rahayu, WJ & Dillon, D 2006 (in pint), *Object-Component based Modeling and Design*.

Dillon, TS & Tan, PL 1993, *Object-Oriented Conceptual Modeling*, Prentice Hall, Australia.

Elmasri, R & Navathe, S 2004, *Fundamentals of database systems*, 4th edn, Pearson/Addison Wesley, New York.

Fankhauser, P & Klement, T 2003, 'XML for Data Warehousing Chances and Challenges (Extended Abstract)', *International Conference on DaWaK '03*, Springer-Verlag GmbH, Prague, Czech Republic.

Feng, L, Chang, E & Dillon, TS 2002, 'A Semantic Network-based Design Methodology for XML Documents', *ACM Transactions on Information Systems (TOIS)*, vol. 20, No 4, no. 4, pp. 390 - 421

Feng, L, Chang, E & Dillon, TS 2003, 'Schemata Transformation of Object-Oriented Conceptual Models to XML', *International Journal of Computer Systems Science & Engineering*, vol. 18, No. 1, no. 1, pp. 45-60

Feng, L & Dillon, TS 2005, 'An XML-Enabled Data Mining Query Language XML-DMQL', *International Journal of Business Intelligence and Data Mining*,

Feng, L, Dillon, TS, Weigand, H & Chang, E 2003, 'An XML-Enabled Association Rule Framework', *14th International Conference on Database and Expert Systems Applications (DEXA '03) 2003*, vol. 2736, ed. WR Vladimír Marík, Olga Stepánková, Springer-Verlag Heidelberg, Prague, Czech Republic, pp. 88 - 97.

Graham, I, Wills, AC & O'Callaghan, AJ 2001, *Object-oriented methods : principles & practice*, 3rd edn, Addison-Wesley, Harlow.

Humphris, M, Hawkins, MW & Dy, MC 1999, *Data Warehousing: Architecture & Implementation*, Prentice Hall PTR, USA.

Inmon, WH, Imhoff, C & Battas, G 1996, *Building the operational data store*, John Wiley & Sons, New York, USA.

Jeong, E & Hsu, C-N 2001, 'Introduction of Integrated View for XML Data with Heterogenous DTDs', *Proceedings of the 10th International Conference on Information and Knowledge Management (CIKM '01)*, ACM, Atlanta, Georgia, pp. 151-8.

Kim, W & Kelly, W 1995, 'Chapter 6: On View Support in Object-Oriented Database Systems', in *Modern Database Systems*, Addison-Wesley Publishing Company, pp. 108-29.

Kimball, R & Ross, M 2002, *The data warehouse toolkit : the complete guide to dimensional modeling*, 2nd edn, Wiley, New York.

Lucie-Xyleme 2001, 'Xyleme: A Dynamic Warehouse for XML Data of the Web', *International Database Engineering & Applications Symposium (IDEAS '01)*, eds ME Adiba, C Collet & BC Desai, IEEE Computer Society 2001, Grenoble, France, pp. 3-7.

Medina, E, Luján-Mora, S & Trujillo, J 2002, 'Handling Conceptual Multidimensional Models Using XML through DTDs', *Proceedings of the 19th British National Conference on Databases: Advances in Databases*, vol. 2405, Springer, pp. 66 - 9.

MIX 1999, *Mediation of Information using XML, (http://www.db.ucsd.edu/projects/MIX/)*.

OASIS 2005, *OASIS Cover Pages (http://www.oasis-open.org/)*, OASIS.

OMG-MDA 2003, *The Architecture of Choice for a Changing World®, MDA Guide Version 1.0.1 (http://www.omg.org/mda/)*, OMG, 2005.

OMG-MOF™ 2003, *Meta-Object Facility (MOF™), Ver 1.4 (http://www.omg.org/technology/documents/modeling_spec_catalog.htm#MOF)*, OMG, 2005.

OMG-OCL 2003, *UML 2.0 OCL Final Adopted specification (http://www.omg.org/cgi-bin/doc?ptc/2003-10-14)*, OMG, 2005.

OMG-UML™ 2003, *UML 2.0 Final Adopted Specification (http://www.uml.org/#UML2.0)*, OMG, 2005.

Ponniah, P 2001, *Data Warehousing Fundamentals: A Comprehensive Guide for IT professionals*, John Wiley & Sons Inc., NY.

Rahayu, JW 2000, 'Object-Relational Transformation Methodology', Doctor of Philosophy (Ph.D) thesis, La Trobe University, Melbourne, Australia.

Rumbaugh, J, Booch, G & Jacobson, I 2004, *The unified modeling language reference manual*, 2nd edn, Addison-Wesley, Boston.

SPARQL 2004, *SPARQL Query Language for RDF, (http://www.w3.org/TR/2005/WD-rdf-sparql-query-20050217/)*, W3C.

Steele, R, Gardner, W, Chandra, W & Dillon, TS 2005a, 'Framework & Prototype for Secure XML-based Electronic Medical Records System', *International Journal of Electronic Healthcare*,

Steele, R, Gardner, W, Chandra, W & Dillon, TS 2005b, 'XML Repository Searcher-Browser Supporting Fine-Grained Access Control', *International Journal of Computers and Application*,

Tan, H, Dillon, TS, Hadzic, F & Feng, L 2005, 'MB3-Miner: mining eMBedded subTREEs using Tree Model Guided candidate generation', *The 1st International IEEE Workshop on Mining Complex Data (MCD '05), In conjunction with ICDM '05)*, IEEE Computer Society, Houston, Texas, USA.

W3C-DTD 2001, *XML Schema (http://www.w3.org/*, W3C, 2005.

W3C-NS 1999, *Namespaces in XML (http://www.w3.org/TR/REC-xml-names/)*, The World Wide Web Consortium (W3C).

W3C-OWL 2004, *Web Ontology Language (OWL), (http://www.w3.org/2004/OWL/)*, The World Wide Web Consortium (W3C), viewed March 2005 http://www.w3.org/2004/OWL/.

W3C-RDQL 2004, *RDQL - A Query Language for RDF, (http://www.w3.org/Submission/2004/SUBM-RDQL-20040109/)*, W3C.

W3C-XML 2004, *Extensible Markup Language (XML) 1.0, (http://www.w3.org/XML/)*, The World Wide Web Consortium (W3C).

W3C-XQuery 2004, *XQuery 1.0: An XML Query Language (http://www.w3.org/TR/xquery)*, The World Wide Web Consortium (W3C), viewed November 2003.

W3C-XSD 2001, *XML Schema (http://www.w3.org/XML/Schema)*, W3C, 2004.

Wouters, C 2006, 'A Formalization and Application of Ontology Extraction', Doctor of Philosophy (Ph.D) thesis, La Trobe University, Melbourne, Australia, Melbourne.

Wouters, C, Dillon, TS, Rahayu, JW, Chang, E & Meersman, R 2004, 'A Practical Approach to the Derivation of a Materialized Ontology View', in D Taniar & W Rahayu (eds), *Web Information Systems*, Idea Group Publishing, USA.

Xiaou, R, Dillon, TS, Chang, E & Feng, L 2001a, 'Mapping Object Relationships into XML Schema', *Proceedings of OOPSLA Workshop on Objects, XML and Databases.*

Xiaou, R, Dillon, TS, Chang, E & Feng, L 2001b, 'Modeling and Transformation of Object-Oriented Conceptual Models into XML Schema', *12th International Conference on Database and Expert Systems Applications (DEXA '01) 2001*, vol. 2113, ed. JL Heinrich C. Mayr, Gerald Quirchmayr, Pavel Vogel, Springer.

Chapter 4

Overview of the Solution

4.1 Introduction

The previous chapter clearly defines the problem addressed in this thesis. This chapter gives an overview of the problem addressed in this thesis. In the following chapters (Chapters 5, 6 & 7), we will focus in detail on certain aspects of the overall problem, in a precise and comprehensive manner.

In this chapter, before we continue with a discussion of our research, we first describe two case study examples that are used in this research to illustrate our concepts in detail. These are given in section 4.2. In section 4.3, we briefly look at the proposed solution followed by section 4.4 that informally describes an XML view document. Sections 4.5-4.10 progressively outline the view concepts, properties, types and view hierarchy followed by section 4.11, which provides a brief discussion of the view design methodology that includes modelling and transformation methodologies adopted in this research. Section 4.12 provides a brief overview of the applications of the proposed XML views in real-world scenarios. This is followed by the summary and the conclusion of this chapter in section 4.13.

4.2 An illustrative Case-Study Examples

To illustrate our theory and concepts, we use two real-world example case studies in this research, namely:

1. simple Conference Publishing System (CPS); and
2. a e-Logistics and Warehouse Management System (e-Sol)

We use these two case studies as XML can be either *document-centric* or *data-centric* (or it can be a mixture of both sorts, such in the case of web data/documents). In our examples, the CPS case study is document-centric in nature and the e-Sol is data-centric. Thus, using these two types, we show that our concepts are capable of being applied to both document-centric and data-centric XML domains.

In this chapter and in the following chapters (5, 6, & 7) we gradually build our concepts and notations using these example case studies.

4.2.1 Illustrative Case-Study Example - I

As an example to illustrate our concepts with document-centric XML domains, we investigate a possible conference publishing system (CPS), for managing and distributing conference proceedings for various international conferences held in different cities throughout the year. The main components of a conference publication system comprises of a collection of papers (past and present), stored in various geographically distributed conference databases/systems, in varying proceedings formats such as ACM, LNCS, IEEE etc. The system is similar to those of existing systems such as ACM Portal (ACM-Portal 2005), SpringerLink (SpringerLink 2005) or IEEE Xplore® (IEEE-Xplore 2004). Logically, we treat all the different conferences and their proceedings as one big (logical) conference proceeding on the web. A simplified UML representation of this case study is given Fig. 4.1 followed by Fig. 4.2 that shows the XSemantic representation of the CPS (using the notations described in Chapter 3).

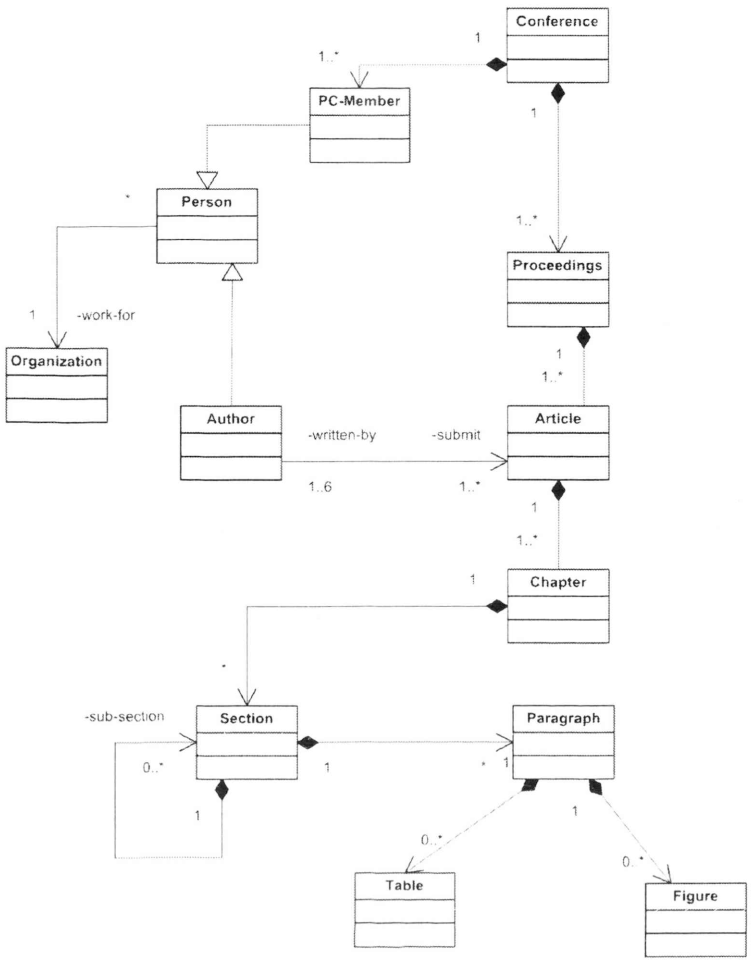


Figure 4.1: CPS simplified model (UML)

4.2.2 An illustrative Case-Study Example - II

The e-Sol Inc. aims to provide logistics, warehouse, and cold storage space for its global customers and collaborative partners, as illustrated in the context diagrams in Fig. 4.3. The e-Sol solution includes a standalone and distributed Warehouse Management System (WMS/e-WMS), and a Logistics Management System (LMS/e-LMS) on an integrated e-Business framework called e-Hub (Elizabeth Chang et al. 2003) for all inter-connected services for customers, business customers, collaborative partner companies, and LWC staff (for e-commerce B2B and B2C). Some real-world applications of such a company, its operations and IT infrastructure can be found in (Elizabeth Chang et al. 2003; Elizabeth Chang et al. 2001; ITEC 2002). For ease of understanding, a simplified use-case diagram of the system is provided in Fig.4.4.

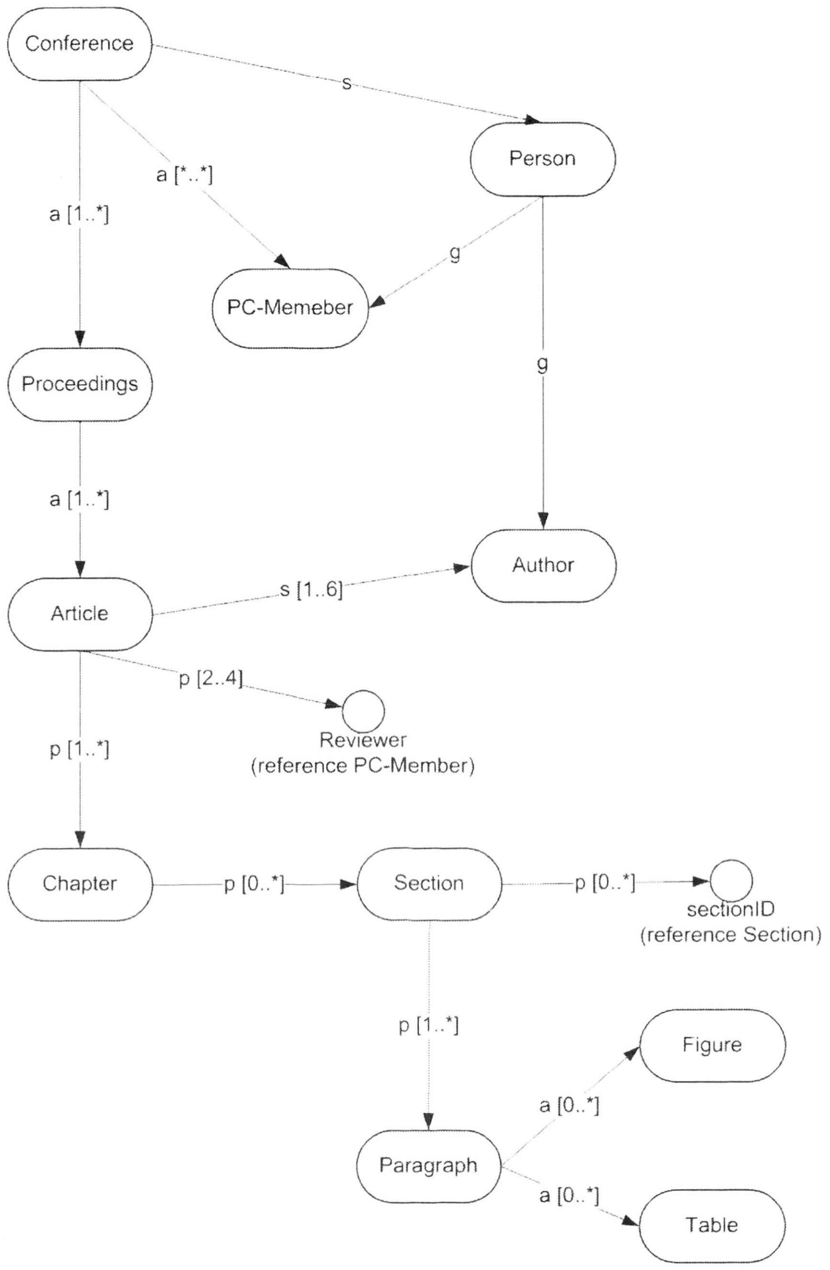


Figure 4.2: CPS simplified model (XSemantic nets)

In the WMS, customers book/reserve warehouse and cold storage space for their goods. They send in a request to warehouse staff via fax, email, or phone, and depending on warehouse capacity and customer type (individual, company or collaborative partner), they get a booking confirmation and a price quote. In addition, customers can also request additional services such as logistics, packing, packaging etc. When the goods physically arrive at the warehouse, they are stamped, sorted, assigned lot numbers and entered into the warehouse database (in Lots-Master). From that day onwards, customers get regular invoices for payments. In addition, customers can ask the warehouse to handle partial sales of their goods to other warehouse customers (updates Lots-Movement and Goods-Transfer), sales to overseas (handled by LMS) or take out the goods in full or in partial (Lots-Movement).

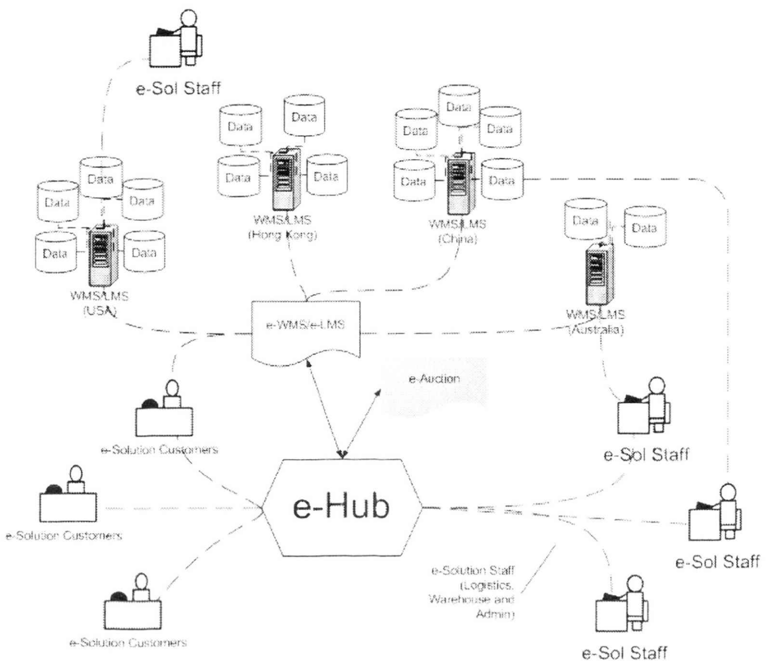


Figure 4.3: e-Sol context diagram (simplified)

Also, customers can check, monitor their lots, buy/sell lots and pay orders via an e-Commerce system called e-WMS. In LMS, customers use/request logistics services (warehouse or third-party logistics providers) provided by the warehouse chains. This service can be regional or global including multi-national shipping companies. Like e-WMS, e-LMS provide customers and warehouses with an e-Commerce based system with which to do business. In e-Hub, all warehouse services are integrated to provide one-stop warehouse services (warehouse, logistics, auction, goods tracking, payment etc) to customers, third-party collaborators and potential customers.

In e-Sol, due to the business process, data has to be in different formats to support multiple systems, customers, warehouses and logistics providers. Also, data has to be duplicated at various points in time, in multiple databases, to support collaborative business needs. In addition, to allow new customers/providers to join the system (or leave), the data formats have to be dynamic and should be efficiently duplicated without loss of semantics. This presents an opportunity to investigate how to utilize the XML conceptual, schema and instance views to design e-Sol at a higher level of abstractions to support changing business, environments, and data formats.

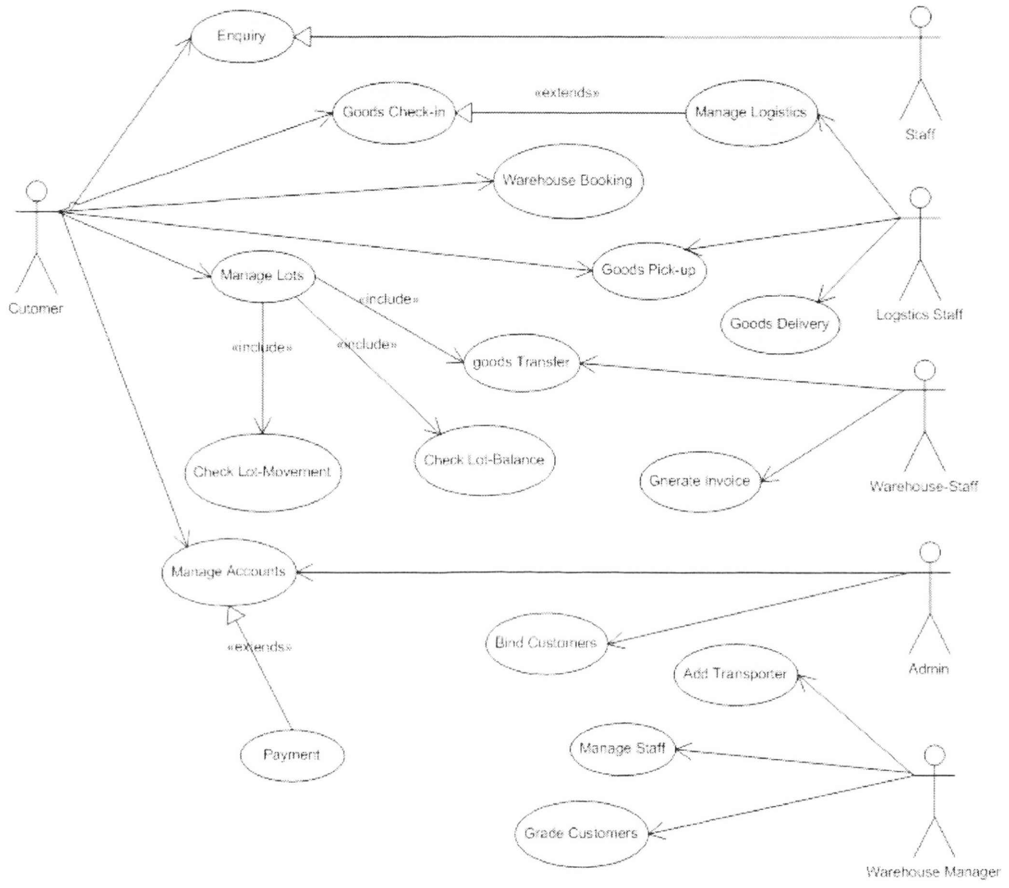


Figure 4.4: A simplified e-Sol use-case diagram

4.3 Solution Outline

In this research, we first look at the semantics of an XML view, referred to as an *imaginary XML document*.[1] In this research, an imaginary XML document is equivalent to the notion of an XML view document that is well-formed and valid against an XML view schema. Section 4.4 provides detailed discussions of such imaginary XML documents.

[1] One could choose "virtual XML document" to conform to a view as a "virtual table" or a "virtual class" in relational or OO systems. However, the use of a virtual XML document has already been used to provide other meanings. Hence, our choice of imaginary XML documents.

Secondly, we look at the *conceptualizations* of such views, in order to separate the conceptual semantics and the implementation semantics. However, since there exist no formal semantics, notations or representations to define or specify views with conceptual semantics, we introduce a new notion of *conceptual views*: a view representation to express views using a conceptual model. Conceptual views are views with conceptual semantics and are free of implementation specific semantics such as query language specific expressions, model specific constraints, etc. We first clearly define and provide formal semantics, properties, constraints and representational semantics for conceptual views. Section 4.5 looks at conceptual views in detail and their formal semantics are given in Chapter 5.

Conversely, since a view mechanism should provide a meaningful view construction formalism or method, here in this thesis, in order to construct conceptual views, we introduce the concept of *conceptual operators*. Thus, conceptual operators are the third part of the overall solution of this research. For example, in the relational model, relational operators operate on one or more relations. Thus, a relational view definition may be comprised of one or more such relational operators to extract and elaborate data from one or more stored relations. Therefore, conceptual operators are comparable to such operators. However, a conceptual operator is a basic procedure that operates on conceptual artefacts (e.g. UML classes, relationships, etc.) in comparison to physically stored (instance) relations or objects. In this research, we propose five binary and four unary operators that form the "complete set" of operators. That is, by combining these seven basic operators in a given sequence, other complex operators, such as division operators (Elmasri & Navathe 2004; Ullman 1989) or compression operators (Wouters 2006) can be derived. Section 4.6 looks at conceptual operators in some detail and their formal semantics are given in Chapter 5.

The fourth part of this research looks at providing a framework for modelling and designing XML views that utilizes the Object-Oriented conceptual modelling paradigm and carries out automated transformation to get:

(i) A schema for the XML view (using XML Schema), which is referred to as logical or schema view that is described in section 4.7. An (XML) view schema is an XML Schema that validates, describes and constrains an XML view document.

(ii) View construct (using a XML query language, e.g. XQuery), which is referred in this thesis as a document or instance view, which is described in some detail in section 4.8. A (XML) document view is a query expression that extracts and elaborates tree items from the corresponding stored XML document tree/(s), as specified by the conceptual operator.

The formal semantics of logical and document views are also given in Chapter 5.

Another reason to have a well defined automated transformation technique is to support, existing and emerging new standards for XML data (semantics, representation, querying, etc.), as they are constantly evolving and need to co-exist with old standard/(s). This is one of the challenges for an XML view mechanism, if that is strongly coupled to one or more specific standards. Thus, having view definitions automatically transformed to a desired technological standard (here, XML Schema and XQuery), at the required level of abstraction are a few of the benefits of having XML views defined:

(a) at a higher level of abstraction;

(b) separated from implementation semantics; and

(c) using formal semantics and notations.

The proposed design and transformation framework is discussed in section 4.11 below.

Finally, the practical value of a concept is as important as the concept itself. Therefore, as the last part of the solution is validation, that is, we investigate the utilization of the proposed XML views in real-world scenarios. Our aim is to provide solution frameworks (or called *architectural frameworks* as referred to in this research), using the proposed XML views in the domains such as XML document warehouses, Semantic Web (SW) (W3C-SW 2005) paradigm and web engineering (Fig. 4.9). An outline is provided in Section 4.12 below.

4.4 Views for XML

As we have stated, our approach to an XML view is based on an XML view mechanism which provides an abstract definition of a view at the early stages of analysis and design, independent of the underlying storage/physical schema. To do so, we provide a view definition at the conceptual level that captures user requirement/perception. This conceptual view can then be mapped to an XML view document.

4.4.1 Imaginary Document Terminology

As mentioned in Chapter 2, Table 4.1 summarises view terminologies using different paradigms, at different levels of abstraction, namely conceptual logical and instance levels.

Table 4.1: View Terminology for Different Paradigms

Level of Abstraction / Paradigm	**Conceptual Level**	**Logical/Schema Level**	**Instance Level**
Relational paradigm	- n/a -	virtual relation	tuple
OO Paradigm	- n/a –	- n/a -	virtual object or imaginary object
XML	- n/a -	*optional* (view) DTD or XML Schema	instance XML document

In relational databases, view is defined as a *virtual relation*,(Codd 1983; Date 2003) and as a *virtual class* in OO databases. But it should be noted that, though views in OO are defined as virtual classes, there exists neither a conceptual level nor a schema level representations. But work such as (Abiteboul & Bonner 1991), at the instance level, distinguishes between: (a) virtual objects: i.e. objects that are derived from existing *stored* instance objects and are logically grouped into a virtual class; and (b) imaginary objects, i.e. *new* objects that are instances of a virtual class but do not have corresponding stored instance objects.

In the context of XML, authors such as Arbiteboul et al. (Abiteboul 1999) carried forward the OO view notion to XML. Later, as described in Chapter 2, the XML view notion is presented in a synonymous manner to the OO views, but may have optional DTD to XML Schema to describe and constrain the instance XML document (Cluet, Veltri & Vodislav 2001). This is because, an XML view, due to its static nature may be considered as an OO type structure by some areas of research.

In this research, as stated in section 4.4 above, to enable separation of conceptual, logical and implementation semantics (i.e. conceptualization) for the XML views, we allow XML views to have three levels of abstraction with corresponding view representations at each level of abstraction, as shown in Table 4.2.

Therefore, we state an XML view as an **imaginary XML document**. Imaginary XML documents are represented using: (a) imaginary classes at the conceptual level; (b) imaginary document types at the schema level; and (c) imaginary instance documents at the instance level.

We use the term *imaginary* document rather than *virtual* document to differentiate the widely accepted norm of referring to any electronic and/or dynamic online documents as *virtual* documents. In this thesis, we use the terms imaginary XML document and XML view document interchangeably. They all refer to the same concept: the XML view document.

Also, it should be noted that imaginary classes and objects are used in a different context in the references (Abiteboul 1999; Abiteboul & Bonner 1991). In this research, imaginary classes correspond to the conceptual level representation of XML views.

Table 4.2: View Terminology/Representation for the Different Paradigms

Level of Abstraction / Paradigm	**Conceptual Level**	**Logical/Schema Level**	**Instance Level**
Relational paradigm	- n/a -	virtual relation	tuple
OO Paradigm	- n/a –	- n/a -	virtual object or imaginary object
XML	***imaginary class***	***imaginary document type***	***imaginary instance document***

4.4.2 Imaginary XML Document

A valid imaginary XML document is an XML view document, which is of correct (XML) syntax, well-formed (W3C-XML 2004) and is constrained by a valid XML (view) schema (W3C-XSD 2001). Also, it is a document that behaves as any other *stored* document that is a valid and well-formed XML document. It may be dynamically created and/or deleted, support queries issued against it, and other views may be defined over it. An imaginary XML document may also support presentation semantics and presentation constraints that may have applied to its view definition.

The XML view mechanism we propose includes the following features.

- View definition and specification is also possible at varying levels of abstraction, namely, conceptual, logical (or schema) and document (or instance level). Thus, the view mechanism discussed in this thesis is referred to, hereafter, as the "*Layered View Model for XML*"

- The view specification is done at the highest level of abstraction: at the *conceptual level*. Our notion of XML view is that it is defined at the conceptual level using abstract notations, such as classes, entities and constraints rather than as a named query expression. Another advantage of having the view definition at this higher level of abstraction helps to avoid the loss of conceptual semantics during the implementation stage as it gives flexibility in defining views at a high level of abstraction, unconstrained by a specific data manipulation language.
- The definition of views at the conceptual level helps the views to be more responsive to both domain and schema level changes as well as to document level changes. Also, due to its abstract nature, the view definitions are portable and generic in nature.
- The view mechanism supports a visual view design methodology using high level Object-Oriented (OO) conceptual modelling languages. It should be noted here that this is different from existing works such as (Braga, Campi & Ceri 2005), where visual view design is done using Query-by-Example tools.
- The layered view model accompanies the proven OO conceptual modelling and design methodology using the top-down approach. The transformations between the different layers of abstraction are well-defined and proven to avoid loss of conceptual semantics (Feng, Chang & Dillon 2002; Feng, Chang & Dillon 2003; Xiaou et al. 2001a; Xiaou et al. 2001b).
- The schemata (and other) transformations used in the layered view model are automated. That is, the transformations are precisely defined and programmable to provide the mapping between the layers of abstraction.
- In the context of MDA (OMG-MDA 2003), the layered view model is capable of producing PIMs and PSMs at required levels of detail and abstraction (or definitions).

Looking from a modelling perspective, as shown in Table 4.2, the imaginary XML document has three main levels of abstraction. At the highest level of abstraction (i.e. conceptual level), an imaginary document is represented as an imaginary XML class called a *conceptual view*, while in the middle level of abstraction; that is at the logical level, it is represented using an imaginary document type that is referred to as the *logical (or schema) view*. At the lowest level of abstraction, which is at the document (or instance) level, it is represented as an imaginary instance document and called *document or (or instance) view*. Collectively, these three levels of abstraction provide the three core components of the imaginary XML document, in the view model. They are given in Table 4.3.

Table 4.3: Imaginary XML Document Terminology and Representations at Different Abstraction Levels

Level of Abstraction / View	**Conceptual Level**	**Logical/Schema Level**	**Instance Level**
XML view component	conceptual view	logical view	document view
Terminology	imaginary class	imaginary document type	imaginary instance document
Representation	class (e.g. UML class)	XML Schema	XML / XQuery / XQueryX

A conceptual view is a *first-class citizen* in the conceptual model. For example, it may be represented in an OO conceptual model using classes, attributes, methods, constraints etc., similar to other stored domain classes. It may have semantic relationships with other view classes and may form new class hierarchies. It is important to treat the class hierarchies of imaginary classes as being separate from the hierarchies of the domain conceptual model (*see* Section 4.9). One of the most important components of the conceptual views is their view constructs (or commonly referred in this thesis as *conceptual operators*). Conceptual operators are basic procedures that operate on one or more conceptual artefacts (i.e. classes, objects, attributes etc.).

At the logical (or schema) level, the corresponding imaginary XML document component is an imaginary document type definition represented using the W3C XML Schema language. Here, conceptual view definitions are transformed into schema language expressions. In this research, we use the XML Schema language as it is better suited for representing both semantic and synthetic relationships than DTDs. Also, further schema specific semantics may be specified (e.g. constraints, data, default values, domains etc.) to make the logical view more expressive and meaningful.

At the document level, document views are represented using document view expressions (i.e. query expressions or extraction algorithms similar to that of (Wouters et al. 2002, 2005; Wouters et al. 2004b)). The document views are representative of the conceptual constructs/operators discussed above, but transformed into the query language (or declarative algorithm) specific syntaxes. It should be noted here that document views (expressions) are available only at design and/or at compile time and only the resulting instance view (XML) document is available at run-time.

The following sections discuss these components in detail.

4.5 Conceptual Views

A conceptual view is the conceptual level component of an imaginary XML document. It defines, specifies and represents an XML view document with conceptual semantics, i.e. view description, name, attribute set, simple and complex content description and the associated constraints.

Definition 5.1. A conceptual view is the notion of the view that is defined at the conceptual level with a higher level of abstraction and conceptual semantics.

This implies that the conceptual view should include the following characteristics.

- A conceptual view specification is done using a high-level OO modelling notation (e.g. XSemantic nets, UML, etc.). It serves as a blueprint for the construction of imaginary instance documents.
- A conceptual view is described by its properties (i.e. attributes) and the associated constraints.
- An attribute of a conceptual view may be identical to that of a domain class or it could have a different definition and/or domain. It may also be of derived type from one or more attributes of the domain class(es).
- The conceptual view definition includes a view name, properties, their semantics and their domain, valid *conceptual schema* definition which constrains and validates the view in a given modelling notation, new properties which are derived from other (domain object) properties and a constructor (that is specified at the conceptual level) that defines how the view will be generated or constructed.
- A conceptual view (and its properties) is a result of one (or more) conceptual operation/(s) (e.g. union, selection, projection, etc. as described in Section 4.6 below) on one or more domain class(es).
- Conceptual views may have their own hierarchical relationships (e.g. generalization/specialization, composition, etc.) and the associated semantics. But these hierarchies are kept separate from the domain class hierarchy (*see* Section 4.9).
- A conceptual view V_1^c may be a composite class(es) consisting of components (or part-of) (of conceptual views e.g. $V_{11}^c, V_{12}^c, V_{13}^c, \ldots.$)) where V_1^c is derived from the domain composite class C_1 and $V_{11}^c, V_{12}^c, V_{13}^c, \ldots.$ are derived from the domain classes $C_{11}, C_{12}, C_{13}, \ldots.$ respectively, where C_1 is a composite domain class with $C_{11}, C_{12}, C_{13}, \ldots.$ as its component (or part-of) classes.
- Conceptual views may have association relationships with existing domain classes.

In simple terms, a conceptual view describes how a collection of XML tags makes sense to a domain user[2] at the conceptual/abstract level. A typical XML domain may contain only few XML documents to many thousands of semantically related, clusters of XML documents (and their related schemas) depending on the real world requirements. At a given instant, only a subset of this cluster of XML tags, their specification and their data values (information) may be of use or required by a domain user. This subset of XML tags, collectively form a conceptual view which is of interest to the domain user at a given point in time.

In order to illustrate the conceptual view mechanism, we grouped the domain user's view requirements into the following categories, namely:

(i) new user requirements

(ii) user perception

(iii) user trigger

(iv) user query

(v) data refinement/elaboration

(vi) access control

- **New user requirements as conceptual views**

An example of a new user requirement conceptual view is an ADDRESS_BOOK of the CPS, where all persons and their contact details in the CPS are pulled out to form a *conceptual view*. This includes the person's name (first, middle, last), street address, postcode, country, phone numbers and email. These types of *conceptual views* are very useful in modelling dimensional data models, where complex new data definitions are derived from one or more existing data definitions at higher levels of abstraction.

For example, declaratively one can state this as;

[2] Domain User: here we use this term very generally to refer to all people who are working in a particular domain and not to task-specific people.

```
Get all the addresses of ALL (AUTHOR, REFEREE)
in the CPS as an
ADDRESS_BOOK::= (Title, FirstName, LastName,
                 AddressLine, City, Postcode,
                Country)
```

- **User perception as conceptual views**

For example, there may exist a user perception "BOOKS" (independent of how the conceptual model is implemented in the storage model) of the CS, where the individual components of a book (i.e. publication, chapter, section, paragraph, figure, tables) "*assembled*" behave as if they are one composite. Therefore, BOOKS may form a *conceptual view*, independent of the storage/implementation model.

For example, declaratively one can state this as:

```
Organise relevant CHAPTER (Title, Content) and
SECTION (Title, Content) and PARAGRAPH (Content)
and FIGURE (Caption, Content) and TABEL
(Caption, Content) in CPS into
BOOK ::= ARTICLE(CHAPTER (SECTION (PARAGRAPGH,
FIGURE, TABLE)))
```

Another good example for this is user (or application) specific *ordering*. The **order** semantics are frequently explicitly captured in semi-structured data and XML, where based on the position of the tag (or vertex) in the XML document hierarchy, each access path to that tag/vertex is unique. This is not the case in relational (or typical OO models), as a tuple represent a non-hierarchical, flat structure. Therefore, user perception (or requirement) of a certain document *tag ordering* that is different from the original document hierarchy, is another good candidate for a conceptual view.

For example, there exists a requirement to collect articles (e.g. Semantic Web related) in a given conference proceedings (e.g. ER 2005), according to a given *order* (e.g. Theory, Models, Applications, Industrial Case Study). Declaratively one can state this as:

```
Organise PROCEEIDNGS/ConferenceName = "ER 2005"
with all ARTICLE (*) where ARTICLE/Area =
"Semantic Web"
order by
     first ARTICLE/Type="Theory"
     next ARTICLE/Type="Models"
     next ARTICLE/Type="Applications"
     finally ARTICLE/Type=" Industrial Case
Study"
```

- **User trigger as conceptual views**

In the CPS, there may exist a requirement that, based on some change(s) (trigger) in the underlying XML documents, a view has to be constructed.

For example, one can state this as:

```
Re-Build  Publication  List  when  any  of  the
submitted abstract content changed
```

Another is described below as:

```
Build Publication-Abstract-list when paper due
date is 'today'.
```

- **User query as conceptual views**

These types of *conceptual views* are analogous to that of a relational view, where a view is used as shorthand for a frequently executed and/or complex query. In the case of *conceptual view*, this is defined using the higher level syntax of the conceptual modelling language extended with view operators (similar to that of a pseudo code or semantic net) than in a specific data manipulation/query language. For example, in the CS, some of the individual XML document may be Journal, ShortPaper, Author etc.

There may exist a requirement that,

```
Set higher "priority code" to all ShortPaper to be
published in Conference='DEXA 2003'.
```

- **Data refinement/elaboration as conceptual views**

This *conceptual view* can be made more expressive or complex by refining or adding extra requirements.

For example, in the CPS;

```
Get all higher priority code of Publications to be
published in Conference during 2003 academic year".
```

```
Get all Publications of authors who are members of
the Institute 'University of Technology, Sydney,
Australia'.
```

- **Access control**

One of the originally intended purposes of views is access control; i.e. protection of stored data from unauthorised users, uses, modification and updates. Thus, in this context, *conceptual views* are analogous to those of relational views, where a view is used as access control mechanism. But since conceptual views are specified and/or defined at a higher level of abstraction than their relational (and OO) views, it provides support for two kinds of access control mechanisms. They are:

(i) User Access Control (UAC) using views as user data sources: here, views are defined for each user based on his/her access control profile. This is similar to UAC provided by relational database systems that uses views.

(ii) Declarative UAC with conceptual specification of access policy for both schema level and data level access control (Steele, Gardner et al. 2005a, 2005b; Steele, Gardner, R.Rajugan et al. 2005).

Since a conceptual view is an imaginary class, it can be defined as:

Definition 5.2. A conceptual view is a generic grouping of similar imaginary instance documents that have similar properties.

It should be noted that conceptual views can be also defined using: (a) intension, i.e. the delineation of the view structure and the associated constraints and (b) extension, i.e. enumeration of instances or all the instance that exists (all the imaginary instance documents of the imaginary class), as described in (Dillon & Tan 1993). For example, the structure of the conceptual view ADDRESS_BOOK (with its attributes and constraints) constitutes the intension of the concept while the address book entries constitute the extension.

4.6 Conceptual Operators

We propose a set of *conceptual operators* in order to systematically construct the conceptual views. These operators can be easily transformed into query expressions, user defined functions and/or procedures for implementation. This helps one to construct views at the abstract level (implementation independent) without knowing or worrying about target query/language syntax. Also, in designing views for multiple XML storage platforms (e.g. native XML storage systems, relational, file system etc.), conceptual operators can help one to design one view definition for all platforms which are later mapped to a model or platform specific query (or procedural) expressions. It should be noted that the conceptual operator operates on both the intension and extension of a conceptual object (Fig. 4.5).

Definition 5.3. The A conceptual operator is a simple (unary or binary) operator, that operates on conceptual artefacts and produce a result set which is again a valid conceptual artefact.

Note: Here, we refer to a conceptual artefact to include all conceptual level notions such as classes, attributes, relationships, constraints and relationships.

Conceptual operators are grouped into set operators. They are:

(i) Union operator

(ii) Difference operator

(iii) Intersection operator

(iv) Cartesian product operator

(v) Join (natural, conditional) operator

and a set of unary operators namely:

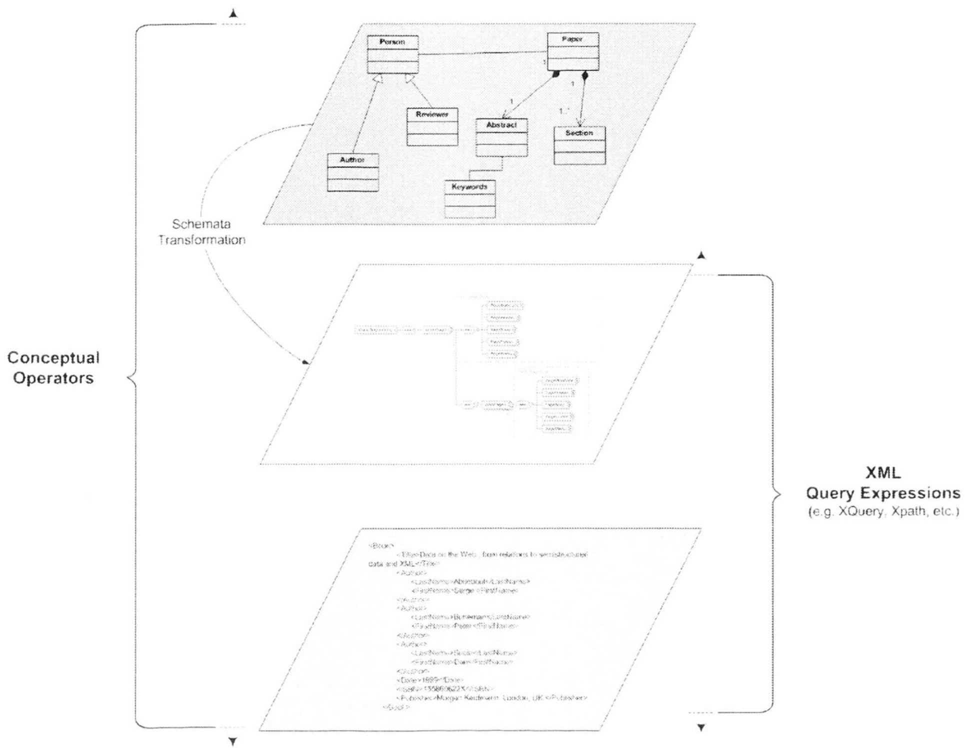
(i) Projection operator

(ii) Selection operator

(iii) Rename operator

(iv) Restructure operator

In Chapter 5, we will provide more formal semantics of such operators and in Chapter 6, a declarative transformation methodology is presented to transform conceptual operators to XQuery (W3C-XQuery 2004) expressions.



```
<Book>
        <Title>Data on the Web : from relations to semistructured
data and XML</Title>
        <Author>
            <LastName>Abiteboul</LastName>
            <FirstName>Serge</FirstName>
        </Author>
        <Author>
            <LastName>Buneman</LastName>
            <FirstName>Peter</FirstName>
        </Author>
        <Author>
            <LastName>Suciu</LastName>
            <FirstName>Dan</FirstName>
        </Author>
        <Date>1999</Date>
        <ISBN>155860622X</ISBN>
        <Publisher>Morgan Kaufmann, London, UK</Publisher>
    </Book>
```

Figure 4.5: Conceptual operators vs. XML query expressions (context diagram)

4.7 Logical (or Schema) Views

A logical (or schema) view is an imaginary document type which describes and constrains imaginary instance documents of an imaginary class. Also, a logical view is automatically derived from the corresponding ***conceptual view*** definition by applying schemata transformations such as those described in (Feng, Chang & Dillon 2002; Feng, Chang & Dillon 2003).

Definition 5.4. A logical view is an imaginary document type that results from a schemata transformation of a corresponding conceptual view definition.

In this research, we use XML Schema to represent imaginary document types (i.e. logical views). Therefore, a logical view is a collection of semantically related (XML) document type declarations and definitions that satisfies a corresponding conceptual view definition from the target. The logical view has the following properties:

- a logical view is an XML schema document. It has a valid name and location as specified in W3C recommendation (W3C-XSD 2001) with a valid root element and a defined namespace;
- the logical view (i.e. the XML schema document) is a result of a well-defined schemata transformation of the corresponding conceptual view definition;
- the logical view describes and constrains the imaginary instance documents of an imaginary class;
- additional constraints (pattern, restriction, scope, default values) can be specified in the logical view to refine the conceptual view definition defined at the conceptual level.

4.8 Document (or Instance) views

A document view is an instantiated imaginary XML document which is an instance of the corresponding imaginary class. It conforms to the logical view defined at the schema level.

Definition 5.5. A document view is an instance of the imaginary class and is described and constrained by the corresponding imaginary document type. It is a valid document that conforms to a corresponding logical view. The instantiation is done by a document view query expression.

Alternatively, definition 5.5 can be stated as: a document view is an instance XML document that satisfies a conceptual view and conforms to a logical view.

An instance view can be used for many purposes such as database views, materialised semantic Web-views, etc., and can be generated from a simple projection of selected document tags to provide dynamic windows into complex heterogenous documents. Based on how these documents are constructed and/or behave, we classify XML instance views into three categories, namely, *derived instance views, constructed instance views,* and *triggered instance views.*

A document view is said to be a valid document (or instance) view, if and only if, it has a valid document name, conforms to a valid (logical view) schema definition (which constrains and validates the document), a collection of semantically related tags and their namespaces (or domain) and (if any) a set of new tags and their namespaces which are derived from others.

Since a document (or instance) view document may result in a few collections of XML tags to that of a whole semantically related cluster of XML tags, for the user, the resulting imaginary view document behaves as another XML document. Thus, it has the following properties:

- A document view is a query expression and instantiates an imaginary instance document at run-time.
- The document view is a result of the transformation of the imaginary class construct (i.e. one or more conceptual operators) to a syntax specific query expression (e.g. XQuery)
- At run-time, a document view includes a valid view name, a valid schema definition which constrains and validates the view, a collection of tags and their namespaces (or domain), new tags (and their namespaces) which are derived from other base domain tags and a constructor that defines how the document will be materialised.
- The document view query expression is any arbitrary query or transformation of the abstract syntax of the conceptual operators in any query language for XML with language specific syntax (e.g. XPath, XQuery) issued against one or more of the stored XML document/(s).

XML instance views can be constructed via an XML query language or some specific algorithms such as Ontology Extraction Methodology (OEM) (Wouters et al. 2004a, 2004b), which can be achieved by the transformation of conceptual operators defined at the conceptual level into a language-specific query fragment, say FLWOR expressions in XQuery, SQL 2003 (ANSI & ISO 2003) or OEM etc. In a related work (*see* Chapter 9), we have shown how conceptual operators can be transformed and mapped to Ontology extraction algorithms.

In this thesis, we discuss XQuery as the document view construct language. Here, we choose XQuery as the document view constructor because it is gaining momentum as the query language of choice for XML databases and repositories. It is a functional language (Chamberlin & Katz 2003) and its functionalities are comparable to early/mid SQL standards (in the relational model). In addition, it is well-suited for our purpose better than other XML query languages (such as XPath or XSLT) as:

(a) XQuery is easy to read and write in comparison to XPath or XSLT;

(b) XQuery expressions, the functional For-Let-Where-Order-Return (FLWOR) expression, XPath/XSLT are purely presentation oriented;

(c) XQuery provides User-Defined Functions (UDF) and variable binding; and

(d) XQuery supports XML Schema.

It should be noted here that, in classical data models (e.g. relational, OO), view materialization and/or construction usually depends on and/or is decided upon using some *a priori* knowledge of the resulting instances (or tuples). This is arguably true, as in such models, the view construct is always an instance-bound, low level construct. However, in our work in LVM, the instance information is (e.g. tuple cardinality, tuple type) is available at higher levels such as at the conceptual and schema levels. Thus, we argue that, it is feasible to decide view materialization and/or construction well before having any a priori knowledge of the document instance. That is, unlike traditional view models, some instance level information is available to the user at a higher level (in the form of class templates, possible instance structure and one or more feasible instances that may satisfy the given schemata, etc.) in the LVM, well before the view is instantiated.

4.9 Imaginary XML Document Hierarchy

As we have stated in Chapter 2, in relational systems, a relation has only scalar valued attributes which are of primitive types. Relations do not have complex hierarchies (specialisation, generalisation, aggregation, composition) (Elizabeth Chang 1996; Kim 1990; Kim & Kelly 1995) with other relations or with virtual relations (views). Therefore, a definition of a relational view (which is part of the external schema (ANSI/SPARC 1978) in regards to other stored relations is straightforward. This is in direct contrast to classes in an OO system.

However, a class in OO systems has both attributes (may be nested or set valued) and methods. A class can also form complex hierarchies (inheritance, part-of, etc) with other classes. Owing to these complexities, definition of views for OO systems is not straightforward. In one of the early discussions on OO data models, Won Kim argues that, an OO data model should be considered as one kind of extended relational model (Kim 1990). Naturally, this statement reflected on early discussions of OO views (Abiteboul & Bonner 1991; Bertino 1992; Elizabeth Chang 1996; Kim 1990; Kim & Kelly 1995). In general, the notion of *virtual class* is considered as the OO equivalent of a view.

In this context, Abiteboul and Bonner 1991 argue that *virtual classes* and stored or *domain classes* are interchangeable in a class hierarchy (Abiteboul & Bonner 1991). They also state that virtual classes are populated by creating new objects (imaginary objects) and from existing objects of other classes (virtual objects).

But Won Kim et al. argue that, in contrast to relational models, class hierarchy and view (virtual) class hierarchy should be kept separate (Kim & Kelly 1995). In continuing this argument, they point out, using several instances (inheritance, nested attributes, schema changes, attribute domains, methods etc.) where, mixing the hierarchy of both classes and views may cause problems including violation of OO semantics (class, relationship, schema, etc.).

In continuing the discussion of view hierarchies, to avoid confusion, here it is imperative to clarify the issue of the relationship between (stored) domain XML documents and the *imaginary XML documents*, that is, the domain class[3] hierarchy and the imaginary class hierarchy.

But, as in the case of OO, it is *not* appropriate to include imaginary classes in the domain class hierarchies and vice versa. This is because:

(i) Selective inheritance problem: an imaginary class may be derived from the domain class with fewer attributes and more methods (Elizabeth Chang 1996). Thus, if one is to allow intermixing of IS-A hierarchies, it is inappropriate to treat the imaginary class as a subclass of the domain class, unless we explicitly allow for selective inheritance. But from database perspective, selective inheritance may cause problems and is discussed in detail in (Rahayu 2000).

(ii) The disjoint partition problem: two or more imaginary classes may be derived from one domain class with different groups of imaginary instance documents. But, one such group of imaginary instance documents may overlap with other group/(s) and non-disjoint (Elizabeth Chang 1996). Thus, though the imaginary classes are derived from the same domain class, their instances (imaginary instance documents) may not disjoint the partition.

(iii) Meaningless class (Bertino 1992; Elizabeth Chang 1996): imaginary classes, though useful, if allowed to participate in the domain class hierarchy may give rise to a semantically meaningless class, to both users and designers, and applications.

(iv) Schema change propagation problem: in OO, schema changes are usually automatically propagated to all the subclasses (and the instances). Thus, if an imaginary class is considered to be part of the

[3] Domain class refers to the conceptual level representation of the stored objects that corresponds to the stored XML documents (i.e. classes in the domain conceptual model).

domain class hierarchy (e.g. subclass), propagation of schema changes could cause problems (Elizabeth Chang 1996; Kim 1990; Kim & Kelly 1995) as such changes may be not appropriate (or violate) for the imaginary class (and imaginary instances document) semantics. This is also true, if a domain class is allowed to participate in the imaginary class hierarchy.

(v) Imaginary schema change problem: If one is to allow domain and imaginary class hierarchies to intermix, there may be a case where, one could have schema changes applied to imaginary classes (i.e. imaginary schema changes) that may result in semantic violation of the stored classes, thus disallowing or forbidding the flexibility of having imaginary schema changes.

(vi) Positioning problem: In the case of allowing for the intermixing of domain and imaginary class hierarchies, it is very difficult to determine the best position for placing an imaginary class appropriately in the existing domain class hierarchy. Also, it is difficult to envision the possible consequences (and effects) of placing an imaginary class inappropriately until it is deployed or at run-time. This is because a wrong placement of an imaginary class in the inheritance hierarchy could lead to the violation of semantics due to the extent of overlapping with an existing class (Elizabeth Chang 1996; Kim 1990; Kim & Kelly 1995).

(vii) Constraint propagation problem: relaxation or restriction of a constraint in the domain (or imaginary) classes that is part of one intermixed class hierarchy may result in loss of semantics and/or violation of OO semantics.

It should be noted that, though we do not address inter-class (methods) and inter-class (messages) dynamics in the context of imaginary classes in this research in detail. Though it is outside the scope of this research to address such dynamic behaviours, some points stated above are applicable to the dynamic properties of the classes (both domain and imaginary). We direct the reader to the works of (Bertino 1992; Elizabeth Chang 1996; Elizabeth Chang et al. 1995; Elizabeth Chang & Dillon 1994; Kim 1990; Kim & Kelly 1995), where comprehensive discussions are provided to address class hierarchies and the problems associated with having intermixed hierarchies, in the context of modelling class behavioural aspects .

Thus, given the points (i) – (vi) above and keeping in accordance with the arguments presented for OO views in (Bertino 1992; Elizabeth Chang 1996; Kim 1990; Kim & Kelly 1995), we believe that the domain (XML) class hierarchy (i.e. stored XML documents) and the imaginary class hierarchy (i.e. imaginary XML documents) should be kept separate. Many of the points made by Won Kim et al. and Chang & Dillon (Elizabeth Chang 1996; Dillon & Tan 1993; Kim & Kelly 1995) for OO views apply to imaginary class as well.

However, *imaginary XML documents* may be allowed to form new document hierarchies (inheritance, nested, composition, etc.), may extend the existing namespace of the stored XML schema namespace and may be used to provide dynamic windows to one or more stored XML documents.

Therefore, at a given instant, in an XML domain, users can add new documents, modify/delete old documents, modify structures/schema of stored documents, create new schemas/hierarchies or create new XML view definitions based on stored documents and/or existing *imaginary XML documents*. But, at any given instant, users *cannot* create new *stored XML document hierarchy* using on existing *XML imaginary XML documents* and vice versa.

4.10 Imaginary Document Classification

XML view can be used for many purposes from simple projection of selected XML tags to providing a dynamic window into complex heterogenous XML documents. Based on how these imaginary documents are constructed, we classify them into three categories, namely Derived Imaginary Document (DID) Constructed Imaginary Document (CID) and Triggered Imaginary Document (TID). The following sections discuss these views and their characteristics.

4.10.1 Derived Imaginary Document (DID)

A *derived imaginary document* is an imaginary document, which is derived from an existing XML document called the *base document*. This is done without violating the original hierarchical structure of the base document. This derivation is similar to that of a XSL transformation, applied on XML documents. The derived documents are as a result of vertical partitioning (projection) or horizontal partitioning (selection) and/or both (including simple join operation within an XML document) of the physical XML documents.

The resulting document is derived when required and only the view definition is persistence in the form of XML schema.

Deduction: DID *is a shallow copy of its original base document.*

This view is very similar to that of SPJ (Select, Project Join) views in relational data model. The difference being, the SPJ operations are done with in a single XML document.

4.10.2 Triggered Imaginary Document (TID)

A *triggered imaginary document* is an imaginary document, where the resulting view document is *triggered* from existing base XML document/(s) and their associated XML schema(s) when a certain "change" or "condition" is satisfied in the XML domain. The trigger which initiates the construction (usually one or more conceptual operators) view is defined as part of the definition, which is unique to TID.

These types of imaginary documents are very useful in defining an *event based* XML documents, where the creation of a document depends on a change in the underlying base XML documents (e.g. user action, value change, state change, version change etc.). The example given above in Section 4.5, point (iii) is a good example of such TID documents.

4.10.3 Constructed Imaginary XML Document (CID)

A *constructed imaginary document* (CID) is an imaginary document, where the resulting XML document is *constructed* from existing one or more stored (i.e. base) XML document(s) and their associated XML schema(s) by manipulating their structure/elements to form a new virtual/abstract imaginary document hierarchy. The construction is done by partitioning (similar to derived XML documents) and/or by re-arranging the XML element information (by hierarchy, order, add, remove etc). The result is a new imaginary document.

This view is similar to that DID, but the base document(s) can be cluster of semantically related XML documents. In other words, CID can satisfy a *conceptual view*, which may span across one or more base documents and/or XML schema(s).

4.11 The Framework: A Layered View Model for XML

The Layered View Model (LVM) for XML is an *architectural framework* for modelling, specifying and transforming XML imaginary documents, at varying levels of abstraction. This is shown in Fig. 4.6. It is a top-down approach and based on the OO conceptual modelling paradigm and transformation methodologies, such as in (Feng, Chang & Dillon 2002; Feng, Chang & Dillon 2003; Xiaou et al. 2001a; Xiaou et al. 2001b), that are specifically developed and extended for XML documents.

As stated previously, the main intention of this architectural construct is to provide a layered view model for XML that is: (i) generic, (ii) valid and well-formed, (iii) query language independent, (iv) auto-generated from high-level view specifications, and (v) provides support for present and emerging query languages.

The accompanying design methodology of the layered view model is elegantly presented in Fig. 4.7. It uses proven Object-Oriented (OO) Conceptual modelling techniques at the top level and has three varying levels of abstraction. In order to clearly explain the problems solved in this thesis, it is imperative that we explain few a concepts and notations that are necessary to follow some of the discussions in this (and the following) chapters.

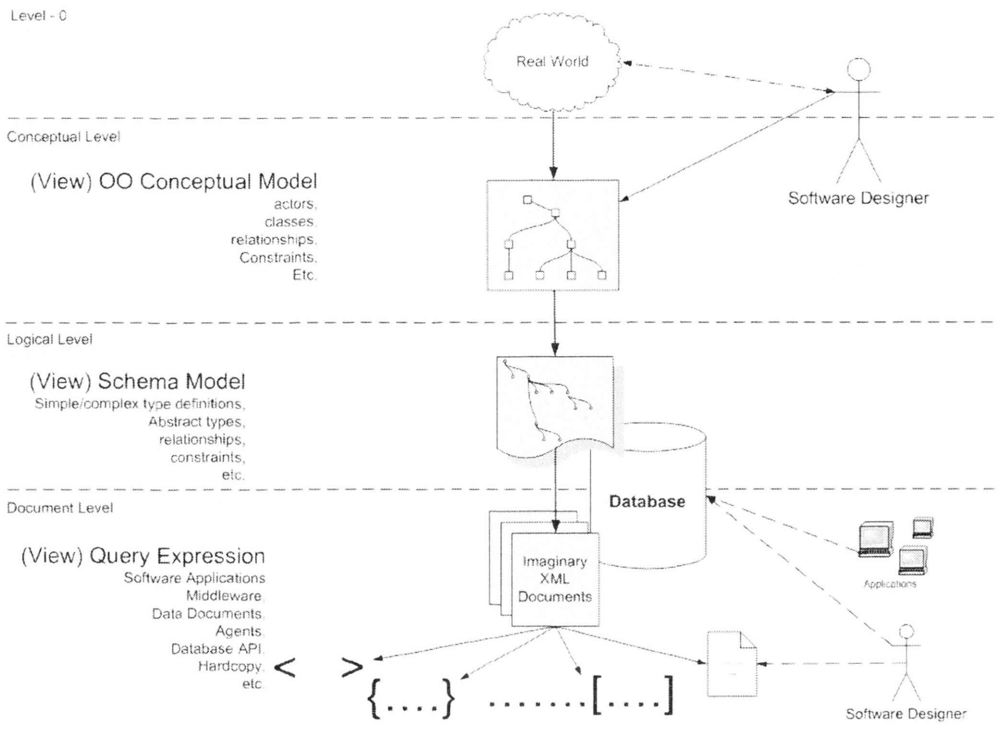


Figure 4.6: An overview of the Layered View Model (LVM)

In software engineering, many methodologies have been proposed to capture real-world problems into manageable representations, which can be communicated, modeled, and developed into robust maintainable software systems. Since the early software models, abstraction and conceptual semantics have proven their importance in software engineering methodologies. For example, in Object-Oriented (OO) conceptual models, they have the power in describing and modelling real-world data semantics and their inter-relationships in a form that is precise and comprehensible to users (Dillon & Tan 1993; Graham, Wills & O'Callaghan 2001). With the emergence of semi-structured data, Semantic Web (SW) (W3C-SW 2005), web services (W3C-WS 2002), and ubiquitous systems, it is important to investigate new data models for Enterprise Information Systems (EIS) that can correlate and co-exist with heterogeneous, multifunctional schemas under the context of Enterprise Content Management (AIIM 2005). Any such data models should accommodate heterogeneous schemas and changing model and data requirements at a higher level of abstraction. In Chapter 3 we defined and described some of the main concepts and notions of OO models used throughout this thesis. Here we look at additional such concepts; the modelling languages and the extensions that are used to model conceptual views in the layered view model.

4.11.1 Modelling Conceptual Views in the Layered View Model (LVM)

In the LVM, as shown in Fig. 4.6 & 4.7, first the software designer models a real-world scenario into a conceptual model using OO conceptual modelling techniques. The conceptual model is preferably represented using one of the high-level (graphical) modelling languages that can represent the real-world objects, concepts and relationships specification at an appropriate level of abstraction. Then s/he proceeds to identify potential views based on the existing user requirements and other model requirements. The designer uses the domain conceptual model (together with interaction and communication with the stakeholders and domain users) as the base for such work. The result of this is a comprehensive and detailed view requirement model that identifies the relevant view requirements.

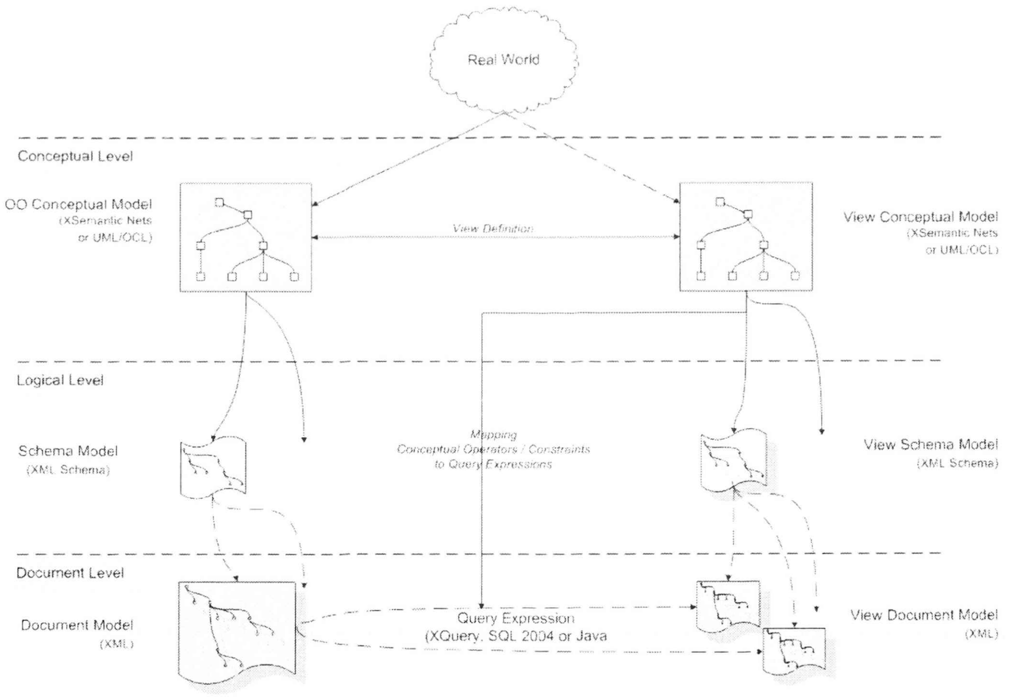


Figure 4.7: The Layered View Model (design steps)

The designer next designs view definitions visually integrating it with the original domain conceptual model. This is one of the unique features of the layered view model, where view definitions are defined at the conceptual level on a par with the domain conceptual model.

It should be noted that this research is different from works such as (Augurusa et al. 2003; Braga & Campi 2003; Braga, Campi & Ceri 2004, 2005; Ceri et al. 2000; Ceri et al. 1999; Oliboni & Tanca 2002), where visual design of views (and queries) are based on instance XML documents and is very similar to that of Query-by-Example tools (Date 2003; Dillon & Tan 1993; Elmasri & Navathe 2004), where views are defined using visual query builders. However, in these works, conceptual and schemata semantics are absent and the view construction is similar to that of writing SQL "CREATE VIEW……….." (Elmasri & Navathe 2004) statements, in the relational model.

In order to define and specify views at the conceptual level (Fig. 4.8), any high-level modelling language that supports OO can be used. For example, Extended-ERD (Elmasri & Navathe 2004) , Dillon & Tan notation (Dillon & Tan 1993), etc. to name a few. In this thesis, we provide discussion of two OO modelling language options, namely: (a) eXtensible Semantic (XSemantic) nets (Feng, Chang & Dillon 2002) and (b) OMG's UML/OCL (OMG-OCL 2003; OMG-UML™ 2003) to demonstrate our concepts. Chapters 8 and 9 provide in-depth discussion of how conceptual views are visually specified using these languages.

As we have stated earlier, in order to systematically construct views from the stored domain objects, in the layered view model, a collection of *conceptual operators* (Fig. 4.5) are used. Unlike the traditional query operators that operate on instance data objects, these operators operate at the conceptual level, on conceptual objects. These operators are also comparable to Query-View-Transformation (QVT) in OMG's Meta-Object Facility (MOF™) (OMG-MOF™ 2003), but focus on conceptual artefacts.

4.11.1.1 The XSemantic Nets

The XSemantic net modelling notion is an intuitive approach to conceptually model XML domains. The modelling efficiency and flexibility comes from its structural similarity to an XML document structure and the ability to capture all the static properties of OO concept; objects, relationships (hierarchical and non-hierarchical) and dependencies to name a few. Here, the concept of nodes and associated constrains are similar or more explicit than the notion of classes in OO models. Also, due to structural similarity, the transformation between XSemantic net and XML is single levelled (i.e. one step) and automatic (Feng, Chang & Dillon 2002). The proposed methodology is comprised of three design levels:

(i) semantic level,

(ii) schema level, and

(iii) instance level.

The aim is to enforce conceptual modelling power to XML (and views) in order to narrow the gap between real-world objects and XML document structures.

The first level corresponds to the Object-Oriented (OO) conceptual level and is composed of two models, namely, the *domain* and the *view* models. This level is based on an (extended) modified semantic network (Feng, Chang & Dillon 2002), referred here as XSemantic nets, that provides semantic modelling of XML domains through five major components, namely:

(i) a set of *nodes* representing real-world *objects* and *view* objects,

(ii) a set of *directed edges* representing *semantic relationships* between these objects,

(iii) a set of *labels* denoting different types of semantic relationships, including

  (a) aggregation,

  (b) generalization,

  (c) association,

  (d) of-property

  (e) view,

(iv) a set of *constraints*,

  (a) defined over a node (e.g. uniqueness, referential integrity, etc.)

  (b) defined over an edge (e.g. cardinality, adhesion, etc.)

(c) defined over a set of edges (e.g. exclusive disjunction, etc.)

(v) a set of *conceptual operators* to systemically construct conceptual views

(a) unary operators

(b) binary operators

(vi) a set of *event nodes* representing *generic* and *user defined* methods (or triggers) in the *view* objects.

The second level of the proposed methodology is concerned with detailed XML schema design for both domain and view objects defined at the semantic level, including *element/attribute declarations* and *simple/complex type definitions*. The mapping between these two design levels are extension of the schemata transformation proposal stated in (Feng, Chang & Dillon 2002) and proposed to transform the semantic models into the XML Schema, based on which XML documents can be systematically created, managed, and validated.

The third level of the design methodology is concerned with a detailed query design for the views defined at the semantic level, including query language specific expressions and syntax declarations. The mapping between semantic level conceptual operators and the query language specific expressions are proposed to transform valid conceptual operators into executable native XML query expressions, such as XQuery FLOWR expressions or SQL 2003/SQLX (ANSI & ISO 2003) statements. The resulting query expressions/statements are able to construct imaginary XML documents that can be validated against the XML (view) schemas generated at the schema level of the design methodology.

At this level, it is also proposed to transform *generic* and *user* defined methods to query expressions in the form of triggers, user defined functions (UDF) and external procedure calls.

4.11.1.2 OMG's UML/OCL

OMG's Unified Modeling Language (UML) (OMG-UML™ 2003) has established itself as the *defacto* modelling language of choice for developing OO conceptual models.

UML provides a well-defined collection of notations, extensions and tools to model a given domain into the needed level of abstraction. It can be said that UML helps to provide a well-defined blueprint of a software system that is easily understood by both users and developers alike. UML also provides extensibility to the modelling language in the form of *stereotypes* which we utilise in our layered view model. Another reason we adopt UML as a view specification language is that, many industrial and academic works such as in (Conrad, Scheffner & Freytag 2000; Feng, Chang & Dillon 2003; Dragan Gaševic et al. 2004b; D. Gaševic et al. 2004; Lujan-Mora, Trujillo & Song 2002; Xiaou et al. 2001a; Xiaou et al. 2001b), have developed (and continue to develop) new MDA, schemata and model transformation methodologies to support modelling and designing applications for the present (e.g. XML, Web Services (W3C-WS 2002), etc.) and emerging (e.g. Semantic Web (W3C-SW 2005), OWL(W3C-OWL 2004), OWL-S. etc.) semi-structured data paradigms.

To model conceptual views in UML, we introduce a set of view specific stereotypes. In addition, to make view constraints more explicit and visible, we use OMG's Object Constraint Language (OCL). In UML, the Object Constraint Language (OCL)(OMG-OCL 2003), which is now a part of the UML 2.0 standard, can support unambiguous constraints specifications for UML models. In our conceptual view model, we incorporate OCL (in addition to built-in UML constraints features) as our view constraint specification language to explicitly state view constraints.

It should be noted that we do not use OCL to define views in our research, rather to express additional constraints using OCL. This is one of the major differences between our research and the work in (Balsters 2003), where OCL is used to define the view construct, where it is being used just as a representation language. Our strong argument for this is:

- OCL should be used to give added precision to the definition of UML (Warmer & Kleppe 2003) artifacts than completely defining the artifacts using it.
- OCL (and other textual declarative syntaxes) should be used as a complementary tool, only to express artifact properties that the UML (and other visual) notations cannot represent.
- OCL is simple, compact and declarative, thus easily adoptable as a complementary language to any modelling notation. Therefore, one concept may be expressed in more than one way (due to its compact declarative and non-mathematical nature), thus may lead to artifact and model ambiguity.
- Since it is declarative in nature, OCL usually states what should be done and not how. Therefore, without a precisely defined based model, it is a challenging and time consuming task to express and precisely define a concept (e.g. a view definition) and is a tedious task to develop algorithms to translate such pure OCL expression (i.e. without base models) to usable program modules.
- OCL by itself may be argued as a modelling language (Balsters 2003), but it is an expression language. It does not provide all the required properties of an OO modelling language (such as ORM (Jarrar, Demey & Meersman 2003), UML or XSemantic nets).
- OCL expressions require a base model (for its types and associations), that is precise in defining its types (i.e. classes, attributes, etc.) or OCL expression may induce further ambiguity and errors.

Thus, in our view model, OCL is utilized as a supplementary expression language, that is used only when visual or modelling language notation was inadequate to represent certain view properties and/or characteristics.

4.11.2 Specifying Conceptual View Constraints in the LVM

In data modelling, specifications often involve constraints. In the case of views, it is usually specified by the data language in which they are defined in. For example, in the relational model, views are defined using SQL and a limited set of constraints can be defined using SQL (Date 2003; Elmasri & Navathe 2004), namely, (1) presentation specific (such as display headings, column width, pattern order etc), (2) range and string patterns for aggregate fields, (3) input formats for updatable views, and (4) other DBMS specific (such view materialization, table block, size, caching options etc).

In Object-Relational and OO models, views have similar constraints but they are more extensive and explicit due to the data model. The views here are constructed and specified by DBMS specific (such as OQL (Cattell et al. 2000)) and/or external languages (such as C++, Java or O_2C (Abiteboul & Bonner 1991)). It is a similar situation in views for the semi-structured data paradigm, where a rich set of view constrains are defined using languages such as OQL based LOREL (Abiteboul et al. 1997). But the work by authors of (Chen, Ling & Lee 2002) provides some form of higher-level view constraints (under ORA-SS model) for XML views, while the work in (Volz, Oberle & Studer 2003) provides some form of logical level view constraints to be defined in views for in SW/RDF paradigm. Here, for our view formalism, we look into using UML/OCL as our view constraint specification language. Also, our work should not be confused with work such as (Balsters 2003), where authors use OCL to define relational views using OCL rather than for constraint specification.

In this research, constraint specification for conceptual views can be explicit and extensive in comparison with relational or OO view constraints as, the conceptual views are designed at a higher-level. Here, the constraint specifications are restricted only by the modelling language semantics and not by the view definition language as in the case of relational (e.g. SQL) and/or OO (e.g. OQL) views. In addition, further constraints can be defined for conceptual views that include:

(i) unique constraints

(ii) referential constraints

(iii) ordered composition

(iv) explicit homogenous composition/heterogeneous compositions,

(v) adhesion and/or dependencies

(vi) exclusive disjunction

(vii) cardinality constraints

(viii) domain constraints (range of values, min, max, pattern etc)

(ix) constructional contents (set, sequence, bag, ordered-set)

In Chapter 5, 6 and 7 we will provide a detailed discussion on these view constraints (with the help of the case study examples and using the extended modelling notations).

4.11.3 View Transformation Methodology in the LVM

After completing the view conceptual model, designers can then apply schemata transformations (*see* Section 4.7) to generate view schemas. These transformations include mapping conceptual elements such as classes, nodes, relationships, constraints etc. to XML Schema types and constructs such as simple/complex type definitions, element, attribute definitions, sequence, order etc.

In the layered view model, there are two main transformation methodologies (Fig. 4.8) used, namely: (i) schemata transformation of conceptual views to logical (or schema) views and (ii) declarative transformation of high-level view constructs (and view methods) to document (or instance) view query expressions.

To transform conceptual view definitions into schema views, in this thesis, we describe two schemata transformation methodologies, namely; (a) XSemantic net (Feng, Chang & Dillon 2002) based transformation methodology and (b) UML/OCL based transformation methodology (Feng, Chang & Dillon 2003; Xiaou et al. 2001a; Xiaou et al. 2001b).

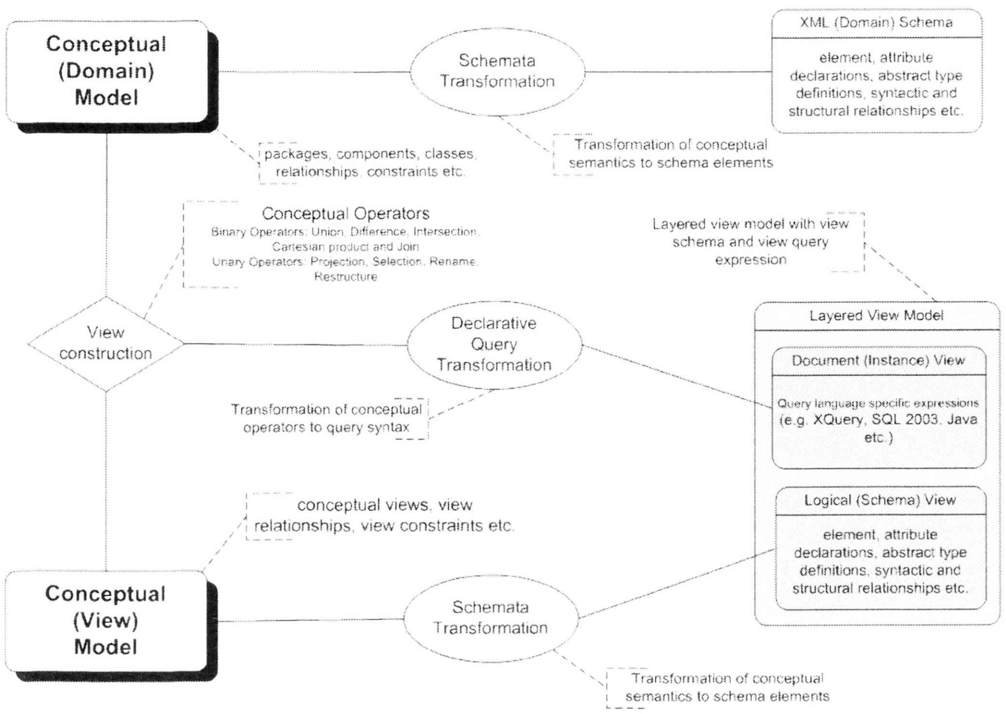

Conceptual (Domain) Model
Schemata Transformation
XML (Domain) Schema
element, attribute declarations, abstract type definitions, syntactic and structural relationships etc.
packages, components, classes, relationships, constraints etc.
Transformation of conceptual semantics to schema elements
Conceptual Operators
Binary Operators: Union, Difference, Intersection, Cartesian product and Join
Unary Operators: Projection, Selection, Rename, Restructure
Layered view model with view schema and view query expression
View construction
Declarative Query Transformation
Layered View Model
Document (Instance) View
Query language specific expressions (e.g. XQuery, SQL 2003, Java etc.)
Transformation of conceptual operators to query syntax
conceptual views, view relationships, view constraints etc.
Logical (Schema) View
element, attribute declarations, abstract type definitions, syntactic and structural relationships etc.
Conceptual (View) Model
Schemata Transformation
Transformation of conceptual semantics to schema elements

Figure 4.8: Transformation methodologies in the LVM

The final step in the design methodology is to apply the declarative transformation methodology to transform view constructs (or conceptual operators) to language specific query expression. In this thesis, we use XQuery as the language of choice as it is gaining momentum as the native query language for XML and provide rich constructs for querying both data-centric and document-centric XML documents.

It should be noted that, the final output of the layered view model is a valid imaginary XML document with a view schema. The encoding of this document, as presented in this thesis is XML, but as shown in Fig. 4.6, the presentation of this imaginary document may vary from pure XML, XHTML, HTML, RDF etc. including Portable Document Format (PDF) documents and printed (paper) format, that is independent of the original XML encoding.

4.12 Designing View Driven Solutions

Designing software constructs using views are an inexpensive way of defining *architectural frameworks* in traditional data engineering approach. Some popular approaches include application wrappers (Mohania, Karlapalem & Kambayashi 1999), data model (Inmon, Imhoff & Battas 1996) or data description wrappers (Gopalkrishnan, Li & Karlapalem 1999), application programming interfaces etc. Since the view definitions are not expensive from a storage point of view and easier (and flexible) to maintain (and in reflecting schema and data changes) than externally maintained application modules, it is well suited for defining virtual data architectures. Also, the conceptual and logical extensions provided in the layered view model provide layered abstraction, data semantics, and constructs (Fig. 4.9) that can be utilized in a systematic manner, similar to the MDA initiatives.

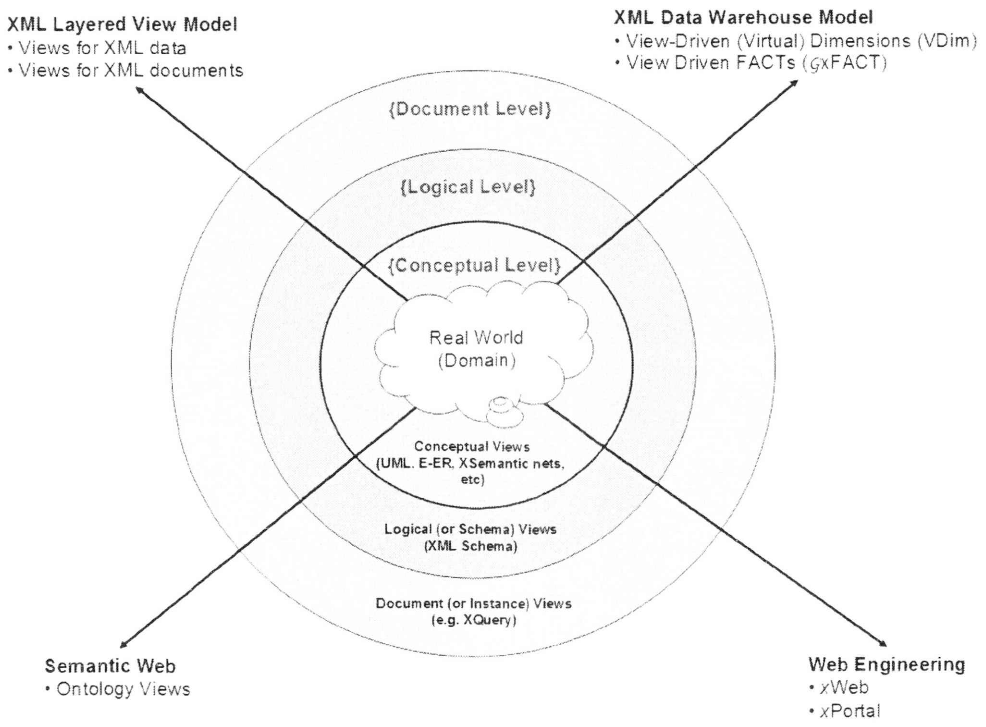

XML Layered View Model
• Views for XML data
• Views for XML documents
XML Data Warehouse Model
• View-Driven (Virtual) Dimensions (VDim)
• View Driven FACTs (*g*xFACT)
{Document Level}
{Logical Level}
{Conceptual Level}
Real World (Domain)
Conceptual Views (UML, E-ER, XSemantic nets, etc)
Logical (or Schema) Views (XML Schema)
Document (or Instance) Views (e.g. XQuery)
Semantic Web
• Ontology Views
Web Engineering
• *x*Web
• *x*Portal

Figure 4.9: Applications and utilization of the LVM and real-world scenarios

Since the introduction of Model-Driven Architecture (MDA) (OMG-MDA 2003) initiative by OMG, platform independent models play a vital role in system development and data engineering. Under the MDA initiative, first the model of a system is *specified* via an abstraction notation that is independent of the technical or deployment specifications (i.e. Platform Independent Model or PIM) and then the PIM is mapped or *transformed* into a deployment model (i.e. Platform Specific Model or PSM) by adding platform or deployment specific information. To support MDA initiatives in data engineering, data semantics, constraints and model requirements have to be specified precisely at a higher level of abstraction.

In the context of MDA solutions for XML domains, it is still a challenging task to produce PIMs despite the flexibility and the semantic richness of the semi-structured schema languages. This is mainly due to OO modelling languages such as OMG UML™, Extended-ER (Elmasri & Navathe 2004) etc. provide insufficient modelling constructs for utilizing XML schema like data descriptions and constraints, while XML Schema lacks the ability to provide higher levels of abstraction (such as conceptual models, visual constraints, etc.) that are easily understood by humans. This is due to the fact that models are often abstract representations which only keep so much of the detail as is relevant to the particular problem being considered (Dillon & Tan 1993; Graham, Wills & O'Callaghan 2001). In this context, XML Schema generally is too low a representation to permit users to interact, visualize or understand it (equivalment of a PIM model). To rectify this situation, many researchers in works such as (Conrad, Scheffner & Freytag 2000; Feng, Chang & Dillon 2002; Feng, Chang & Dillon 2003; Gaševic, Djuric & Devedzic 2005; Dragan Gaševic et al. 2004a), have applied intuitive techniques, notations and transformation methodologies to capture XML (and other semi-structured data) semantics at the conceptual level. This presents an opportunity to investigate data views as a means of providing data abstraction and semantics in PIMs for data intensive MDA solutions, such as XML document warehouses.

Some of the proposed *architectural frameworks* in this thesis are designed using the layered view model (as shown in Fig. 4.9) and described in Chapters 8-10. They are from three different data domains, namely (XML) data warehouse (Chapter 8), Semantic Web (Chapter 9) and website engineering (Chapter 10).

4.13 Conclusion

In this chapter, we provided an overview of the solution proposed in this thesis, i.e. a layered view model (LVM) for XML to solve the problem of views for XML, as described and defined in the previous chapter. The discussion was organised in three logical groups. Firstly, we outlined what is meant by an XML view document in our research. Secondly, we progressively described the unique characteristics of the layered view model and the modelling and transformation issues. Finally, we outlined how the layered view model will be utilized in designing of architectural constructs and solutions for XML domains.

The following chapter, Chapters 5, we will provide a detailed discussion of the LVM concepts, theory and formal properties of views in the LVM. It includes definitions of the concepts used in the LVM, including the theory that it is imperative to understand the views in the LVM. Chapters 6 and 7 demonstrate modelling and transformation of views in the LVM using the illustrative case study example described here in Section 4.2.2. In Chapters 8-10, each chapter describes an application of LVM (as outlined here in Section 4.12).

References

Abiteboul, S 1999, 'On Views and XML', *Proceedings of the eighteenth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems (PODS '99)*, ACM Press New York, NY, USA, Philadelphia, Pennsylvania, USA, pp. 1-9.

Abiteboul, S & Bonner, A 1991, 'Objects and Views', *ACM SIGMOD Record, Proceedings of the International Conference on Management of Data (ACM SIGMOD '91)*, vol. 20 (2), ACM Press New York, NY, USA.

Abiteboul, S, Quass, J, McHugh, J, Widom, J & Wiener, J 1997, 'The Lorel Query Language for Semistructured Data', *International Journal on Digital Libraries*, vol. 1, no. 1, pp. 68-88, April 1997.

ACM-Portal 2005, *(http://portal.acm.org/)*, ACM.

AIIM 2005, *The ECM Association (http://www.aiim.org/index.asp)*, AIIM, http://www.aiim.org.

ANSI & ISO 2003, *ANSI - SQL 2003*, ANSI / ISO.

Augurusa, E, Braga, D, Campi, A & Ceri, S 2003, 'Design and Implementation of a Graphical Interface to XQuery', *ACM Symposium on Applied Computing (SAC '03)*, ACM, Melbourne, USA, pp. 1163-7.

Balsters, H 2003, 'Modelling Database Views with Derived Classes in the UML/OCL-framework', *The Unified Modeling Language: Modeling Languages and Applications (UML '03)*, vol. 2863, Springer, USA, pp. 295-309.

Bertino, E 1992, 'A View Mechanism for Object-Oriented Databases', *3rd International Conference on Extending Database Technology (EDBT '92)*, vol. 580, eds A Pirotte, C Delobel & G Gottlob, Springer, Vienna, Austria, pp. 136-51.

Braga, D & Campi, A 2003, 'A Graphical Environment to Query XML Data with XQuery', *4th International Conference on Web Information Systems Engineering (WISE '03)*, IEEE Computer Society, Rome, Italy, pp. 31-40.

Braga, D, Campi, A & Ceri, S 2004, 'XQBE: A Graphical Interface for XQuery Engines', *9th International Conference on Extending Database Technology (EBT '04)*, vol. LNCS 2992, Springer, Heraklion, Crete, Greece,, pp. 848-50.

Braga, D, Campi, A & Ceri, S 2005, 'XQBE (XQuery By Example): A visual interface to the standard XML query language', *ACM Transactions on Database Systems (TODS)*, vol. 30(2), no. 2, pp. 398-443

Cattell, RGG, Barry, DK, Berler, M, Eastman, J, Jordan, D, Russell, C, Schadow, O, Stanienda, T & Velez, F (eds) 2000, *The Object Data Standard: ODMG 3.0*, Morgan Kaufmann.

Ceri, S, Comai, S, Damiani, E, Fraternali, P & Tanca, L 2000, 'Complex Queries in XML-GL', *Proceedings of the 2000 ACM Symposium on Applied Computing (SAC '00)*, ACM, Como, Italy, pp. 888-93.

Ceri, S, Comai, S, Fraternali, P, Paraboschi, S, Tanca, L & Damiani, E 1999, 'XML-GL: A Graphical Language for Querying and Restructuring XML Documents', *SEBD 1999*, Como, Italy, pp. 151-65.

Chamberlin, DD & Katz, H 2003, *XQuery from the experts : a guide to the W3C XML query language*, Addison-Wesley, Boston.

Chang, E 1996, 'Object Oriented User Interface Design and Usability Evaluation', Doctor of Philosophy (Ph.D) thesis, La Trobe University, Melbourne, Australia.

Chang, E, Dillon, T, Gardner, W, Talevski, A, R.Rajugan & Kapnoullas, T 2003, 'A Virtual Logistics Network and an e-Hub as a Competitive Approach for Small to Medium Size Companies', *2nd International Human.Society@Internet Conference*, Springer-Verlag, Seoul, Korea, pp. 265-71.

Chang, E, Dillon, T, Maher, T & Bloomer, W 1995, *Computer aided Design of User Interfcaces Integrated with Application Software.*

Chang, E & Dillon, TS 1994, 'Integration of User Interfaces with Application Software and Databases Through the Use of Perspectives', *1st International Conference on Object-Role Modeling (ORM '94)*, Australia, pp. 4-6.

Chang, E, Gardner, W, Talevski, A, Gautama, E, R.Rajugan, Kapnoullas, T & Satter, S 2001, 'Virtual Collaborative Logistics and B2B e-Commerce', *e-Business Conference*, Duxon Wellington, NZ.

Chen, YB, Ling, TW & Lee, ML 2002, 'Designing Valid XML Views', *Proceedings of the 21st International Conference on Conceptual Modeling (ER '02)*, vol. 2503, eds S Spaccapietra, ST March & Y Kambayashi, Springer-Verlag London, UK, Tampere, Finland, pp. 463 - 78.

Cluet, S, Veltri, P & Vodislav, D 2001, 'Views in a Large Scale XML Repository', *Proceedings of the 27th VLDB Conference (VLDB '01)*, Roma, Italy.

Codd, EF 1983, 'A relational model of data for large shared data banks', *Communications of the ACM*, vol. 26(1), no. 1, pp. 64-9

Conrad, R, Scheffner, D & Freytag, JC 2000, 'XML conceptual modeling using UML', *19th International Conference on Conceptual Modeling (ER '00)*, vol. 1920, eds AHF Laender, SW Liddle & VC Storey, Springer-Verlag London, UK, USA, pp. 558-71.

Date, CJ 2003, *An introduction to database systems*, 8th edn, Pearson/Addison Wesley, New York.

Dillon, TS & Tan, PL 1993, *Object-Oriented Conceptual Modeling*, Prentice Hall, Australia.

Elmasri, R & Navathe, S 2004, *Fundamentals of database systems*, 4th edn, Pearson/Addison Wesley, New York.

Feng, L, Chang, E & Dillon, TS 2002, 'A Semantic Network-based Design Methodology for XML Documents', *ACM Transactions on Information Systems (TOIS)*, vol. 20, No 4, no. 4, pp. 390 - 421

Feng, L, Chang, E & Dillon, TS 2003, 'Schemata Transformation of Object-Oriented Conceptual Models to XML', *International Journal of Computer Systems Science & Engineering*, vol. 18, No. 1, no. 1, pp. 45-60

Gaševic, D, Djuric, D & Devedzic, V 2005, 'Bridging MDA and OWL Ontologies', *Journal of Web Engineering*, vol. 4(2), no. 2, pp. 118-43

Gaševic, D, Djuric, D, Devedzic, V & Damjanovic, V 2004a, 'Approaching OWL and MDA Through Technological Spaces', *3rd Workshop in Software Model Engineering (WiSME 2004)*, Lisbon, Portugal.

Gaševic, D, Djuric, D, Devedzic, V & Damjanovic, V 2004b, 'Converting UML to OWL Ontologies', *Proceedings of the 13 th International World Wide Web Conference*, NY, USA, pp. 488-9.

Gaševic, D, Djuric, D, Devedžic, V & Damjanovic, V 2004, 'UML for Read-To-Use OWL Ontologies', *Proceedings of the IEEE International Conference Intelligent Systems*, Vrana, Bulgaria.

Gopalkrishnan, V, Li, Q & Karlapalem, K 1999, 'Star/Snow-flake Schema Driven Object-Relational Data Warehouse Design and Query Processing Strategies', *1st First International Conference on Data Warehousing and Knowledge Discovery (DaWaK '99)*, Springer, Florence Italy.

Graham, I, Wills, AC & O'Callaghan, AJ 2001, *Object-oriented methods : principles & practice*, 3rd edn, Addison-Wesley, Harlow.

IEEE-Xplore 2004, *IEEE Xplore®: http://ieeexplore.ieee.org*, IEEE.

Inmon, WH, Imhoff, C & Battas, G 1996, *Building the operational data store*, John Wiley & Sons, New York, USA.

ITEC 2002, *iPower Logistics (http://www.logistics.cbs.curtin.edu.au/)*, http://www.logistics.cbs.curtin.edu.au/.

Jarrar, M, Demey, J & Meersman, R 2003, 'On Using Conceptual Data Modeling for Ontology Engineering', *Journal on Data Semantics*, vol. LNCS 2800, pp. 185-207

Kim, W 1990, 'Research Directions in Object-Oriented Database Systems', *Proceedings of the ninth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, ACM Press New York, NY, USA, Nashville, Tennessee, USA, pp. 1-15.

Kim, W & Kelly, W 1995, 'Chapter 6: On View Support in Object-Oriented Database Systems', in *Modern Database Systems*, Addison-Wesley Publishing Company, pp. 108-29.

Lujan-Mora, S, Trujillo, J & Song, I-Y 2002, 'Extending the UML for Multidimensional Modeling', *Fifth International Conference on the Unified Modeling Language and its applications (UML '02)*, Springer-Verlag London, UK, Dresden, Germany, pp. 290-304.

Mohania, MK, Karlapalem, K & Kambayashi, Y 1999, 'Data Warehouse Design and Maintenance through View Normalization', *10th International Conference on Database and Expert Systems Applications (DEXA '99)*, vol. 1677, Springer, Florence, Italy, pp. 747-50.

Oliboni, B & Tanca, L 2002, 'A visual language should be easy to use: a step forward for XML-GL', *Information Systems*, vol. 27(7), pp. 459-86

OMG-MDA 2003, *The Architecture of Choice for a Changing World®, MDA Guide Version 1.0.1 (http://www.omg.org/mda/)*, OMG, 2005.

OMG-MOF™ 2003, *Meta-Object Facility (MOF™), Ver 1.4 (http://www.omg.org/technology/documents/modeling_spec_catalog.htm#MOF)*, OMG, 2005.

OMG-OCL 2003, *UML 2.0 OCL Final Adopted specification (http://www.omg.org/cgi-bin/doc?ptc/2003-10-14)*, OMG, 2005.

OMG-UML™ 2003, *UML 2.0 Final Adopted Specification (http://www.uml.org/#UML2.0)*, OMG, 2005.

Rahayu, JW 2000, 'Object-Relational Transformation Methodology', Doctor of Philosophy (Ph.D) thesis, La Trobe University, Melbourne, Australia.

SpringerLink 2005, *SpringerLink: http://www.springerlink.com*, Springer.

Steele, R, Gardner, W, Chandra, W & Dillon, TS 2005a, 'Framework & Prototype for Secure XML-based Electronic Medical Records System', *International Journal of Electronic Healthcare*,

Steele, R, Gardner, W, Chandra, W & Dillon, TS 2005b, 'XML Repository Searcher-Browser Supporting Fine-Grained Access Control', *International Journal of Computers and Application*,

Steele, R, Gardner, W, R.Rajugan & Dillon, TS 2005, 'A Design Methodology for User Access Control (UAC) Middleware', *Proceedings of the IEEE International Conference on e-Technology, e-Commerce and e-Service (EEE '05)*, pp. 385-90.

Ullman, JD 1989, *Principles of database and knowledge-base systems*, vol. 1, Principles of computer science series., Computer Science Press, Rockville, Md.

Volz, R, Oberle, D & Studer, R 2003, 'Views for light-weight Web ontologies', *Proceedings of the ACM Symposium on Applied Computing (SAC '03)*, ACM Press New York, NY, USA, USA, pp. 1168 - 73.

W3C-OWL 2004, *Web Ontology Language (OWL), (http://www.w3.org/2004/OWL/)*, The World Wide Web Consortium (W3C), viewed March 2005 http://www.w3.org/2004/OWL/.

W3C-SW 2005, *The Semantic Web (http://www.w3.org/2001/sw/)*, W3C.

W3C-WS 2002, *Web Services Activity, (http://www.w3.org/2002/ws/)*, W3C.

W3C-XML 2004, *Extensible Markup Language (XML) 1.0, (http://www.w3.org/XML/)*, The World Wide Web Consortium (W3C), http://www.w3.org/XML/.

W3C-XQuery 2004, *XQuery 1.0: An XML Query Language (http://www.w3.org/TR/xquery)*, The World Wide Web Consortium (W3C), viewed November 2003.

W3C-XSD 2001, *XML Schema (http://www.w3.org/XML/Schema)*, W3C, 2004.

Warmer, JB & Kleppe, AG 2003, *The object constraint language : getting your models ready for MDA*, 2nd edn, Addison-Wesley, Boston, MA.

Wouters, C 2006, 'A Formalization and Application of Ontology Extraction', Doctor of Philosophy (Ph.D) thesis, La Trobe University, Melbourne, Australia, Melbourne.

Wouters, C, Dillon, TS, Rahayu, JW & Chang, E 2002, 'A Practical Walkthrough of the Ontology Derivation Rules', *Database and Expert Systems Applications : 13th International Conference (DEXA '02)*, Springer, Aix-en-Provence, France, pp. 259-68.

Wouters, C, Dillon, TS, Rahayu, JW & Chang, E 2005, 'Large scale ontology visualisation using ontology extraction', *International Journal of Web and Grid Services*, vol. 1(1), no. 1, pp. 113 - 35

Wouters, C, Dillon, TS, Rahayu, JW, Chang, E & Meersman, R 2004a, 'Ontologies on the MOVE', *9th International Conference on Database Systems for Advanced Applications (DASFAA '04)*, vol. 2973/2004, eds Y Lee, J Li, K-Y Whang & D Lee, Springer-Verlag GmbH, Jeju Island, Korea, pp. 812-23.

Wouters, C, Dillon, TS, Rahayu, JW, Chang, E & Meersman, R 2004b, 'A Practical Approach to the Derivation of a Materialized Ontology View', in D Taniar & W Rahayu (eds), *Web Information Systems*, Idea Group Publishing, USA.

Xiaou, R, Dillon, TS, Chang, E & Feng, L 2001a, 'Mapping Object Relationships into XML Schema', *Proceedings of OOPSLA Workshop on Objects, XML and Databases.*

Xiaou, R, Dillon, TS, Chang, E & Feng, L 2001b, 'Modeling and Transformation of Object-Oriented Conceptual Models into XML Schema', *12th International Conference on Database and Expert Systems Applications (DEXA '01) 2001*, vol. 2113, ed. JL Heinrich C. Mayr, Gerald Quirchmayr, Pavel Vogel, Springer.

Chapter 5

Theory and Concepts

5.1 Introduction

In this chapter, we continue our discussion from Chapter 4 on the layered view model. We focus on the Layered View Model (LVM) concepts, formal semantics, properties and characteristics. This chapter is organized into three parts, namely: LVM theoretical concepts, conceptual operator formal semantics and theory and concepts related to modelling LVM views.

This chapter is organised as follows. First, in section 5.2, the formal semantics of the layered view model are presented including definitions of the concepts that are used to define the layered views. In section 5.3, conceptual operators are discussed in detail including binary operators (section 5.3.1) and unary operators (section 5.3.2). Section 5.4 discusses some of the modelling issues associated with the layered view model, followed by view constraint specification formalism in section 5.5. Section 5.6 concludes this chapter with a summary.

5.2 The Layered View Model: Formal Semantics

As we have presented in Chapter 4, the Layered View Model (LVM) for XML is comprised of three different levels of abstraction (Fig. 5.1), namely:

(i) Conceptual level

(ii) Logical (or schema) level

(iii) Document (or instance) level

The LVM is illustrated in Fig. 5.1.

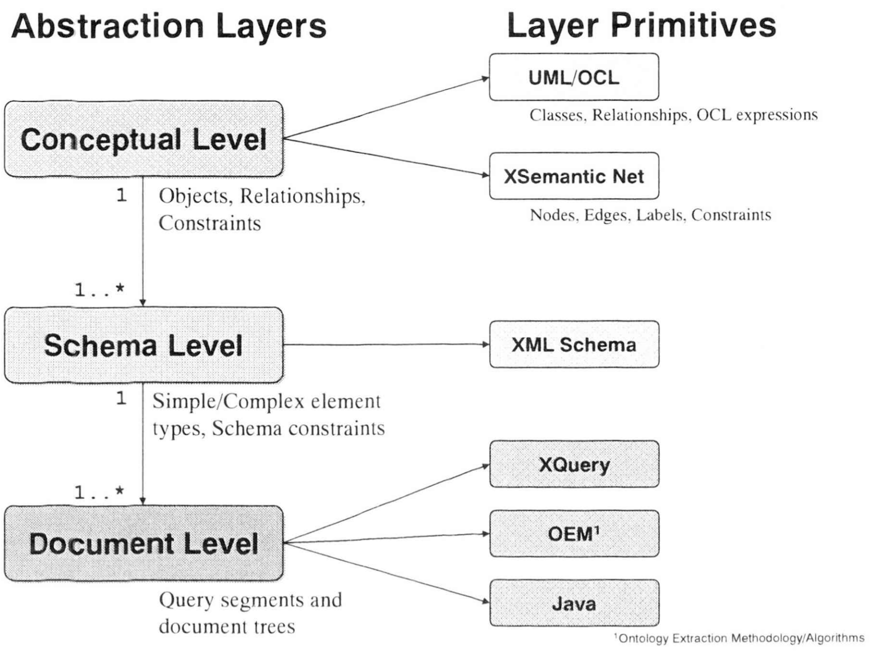

Abstraction Layers
Layer Primitives
Conceptual Level
Schema Level
Document Level
1
Objects, Relationships, Constraints
1..*
1
Simple/Complex element types, Schema constraints
1..*
Query segments and document trees
UML/OCL
Classes, Relationships, OCL expressions
XSemantic Net
Nodes, Edges, Labels, Constraints
XML Schema
XQuery
OEM¹
Java
¹Ontology Extraction Methodology/Algorithms

Figure 5.1: Layered View Model for XML (context diagram)

In this research, XML view study is based on the following two postulates about the real world.

Postulate 5.1. The term ***context*** refers to the domain that interests an organization as a whole. A *context* is more than a measure (Golfarelli, Maio & Rizzi 1998; Trujillo et al. 2001), and implies a meaningful collection of objects, relationships among these objects, as well as some constraints associated with the objects and their relationships that are relevant to its applications.

Let ζ_{ontext} denote a context, such that;

Definition 5.1. A context ζ_{ontext} is a 4-ary tuple of $\zeta_{name}, \zeta_{obj}, \zeta_{rel}$ and $\zeta_{constraints}$, where ζ_{name} is the name of the context ζ_{ontex}, to is a set of objects in ζ_{ontex}, ζ_{rel} is a set of object relationships in ζ_{ontex}, and $\zeta_{constraints}$ is a set of constraints associated with ζ_{obj} and ζ_{rel} in ζ_{ontex}.

$$\zeta_{ontext} = (\zeta_{name}, \zeta_{obj}, \zeta_{rel}, \zeta_{constraints}) \qquad (5.1)$$

In Chapter 3, equation 3.1, we discussed Universe of Discourse (UoD) in detail. Here, using equation 3.1 and equation 5.1 above, we can show that:

$$\forall c_{obj} \in \zeta_{obj} : c_{obj} = (y_1) \ \ with \ \ y_1 \in O_{bject} \qquad (5.2)$$

and

$$\forall c_{rel} \in \zeta_{rel} : c_{rel} = (y_2) \ \ with \ \ y_2 \in R_{elationship} \qquad (5.3)$$

and

$$\forall c_{constraint} \in \zeta_{constraint} : c_{constraint} = (y_3) \ \ with \ \ y_3 \in C_{onstraint} \qquad (5.4)$$

Thus:

$$\forall c_{ontext} \in \zeta_{ontext} : c_{ontext} = (y) \ \ with \ \ y \in UoD \qquad (5.5)$$

Postulate 5.2. The term ***view*** refers to a certain perspective of the context that makes sense to one or more stakeholders of the organization or an organization unit at a given point in time.

Example 5.1: For example, in the CPS as described in Chapter 4, `people`, `paper`, and `order` are examples of a possible context.

Example 5.2: Also, `Address-Book`, `Proceedings-Editor-Book`, `PC-Chair` are two examples of *views* in the context of `people`.

Example 5.3: `Short-Paper` and `Journal-Paper` are examples of views in the context of `Paper`.

5.2.1 Conceptual Level

The top conceptual level (Fig. 5.1) describes the structure and semantics of *imaginary XML documents* in a way which is more comprehensible to human users. It hides the details of implementation semantics and concentrates on visually describing *imaginary classes*, *relationships* among these imaginary classes and the *associated constraints* upon the objects and relationships.

Here, *imaginary classes* are modelled using any visual modelling languages, such as XML-specific XSemantic nets (Feng, Chang & Dillon 2002) (*see* Chapter 6) or UML (OMG-UML™ 2003) (*see* Chapter 7), etc. Thus, the modelling primitives include imaginary classes, attributes, relationships, and constraints. The output of this level is a well-defined *valid* conceptual model in UML, XSemantic net, or even OMG's MOF (Meta-Object-Factory), which can be visual (such as UML class diagrams) with complementary textual notations (in the case of XMI or OCL models).

It is also possible to do such tasks in other visual modelling languages such as ORM[1] or Extended Entity Diagrams (E-ER), but it is not presented in this research.

At this level, a *context* is representative of one or more domain classes represented using modelling primitives like class, attribute, relationship and constraints. To enable the construction of valid conceptual views (i.e. imaginary classes) from such a context, we introduce the notion of *conceptual operators*.

Definition 5.2. A conceptual view V^{c}_{iew} is a 4-ary tuple of $V^{c}_{name}, V^{c}_{obj}, V^{c}_{rel}$ and $V^{c}_{constraints}$, where V^{c}_{name} is the name of the conceptual view V^{c}_{iew}, V^{c}_{obj} is a set of objects in V^{c}_{iew}, V^{c}_{rel} is a set of object relationships in V^{c}_{iew}, and $V^{c}_{constraints}$ is a set of constraints associated with V^{c}_{obj} and V^{c}_{rel} in V^{c}_{iew}.

$$V^{c}_{iew} = (V^{c}_{name}, V^{c}_{obj}, V^{c}_{rel}, V^{c}_{constraints}) \qquad (5.6)$$

5.2.2 Conceptual Operators

Conceptual operators are conceptual level operators that operate on conceptual artefacts. They are grouped into set operators, namely; union, difference, intersection, Cartesian product, join and unary operators namely; projection, rename, restructure, selection and can facilitate systematic construction of conceptual views from a context. These conceptual operators can be easily transformed into query expressions, user-defined functions and/or procedures for implementation. By doing so, they help the modeler to capture view constructs at the abstract level without knowing or worrying about query language syntax. The set of binary and unary operators provided here (except intersection and join) are a *complete* or *primitive* set (Elmasri & Navathe 2004; Ullman 1989); i.e. other operators, such as division, join, intersection and

[1] http://www.orm.net/index.html

compression operators (Wouters et al. 2004) can be derived from these complete set of operators. A detailed description and formal semantics of these conceptual operators are presented below in Section 5.3.

Let $\tilde{\lambda}$ be a set of conceptual operators.

Definition 5.3. A V^{c}_{iew} is called a *valid* conceptual view of the context ζ_{ontext}, if and only if the following conditions (i), (ii), (iii) and (iv) are satisfied:

(i) For any object $\forall o^{v}_{obj} \in V^{c}_{obj}$, there exist objects $\exists o^{c1}_{obj}, o^{c2}_{obj}, ... o^{ci}_{obj}, .. o^{cn}_{obj} \in \zeta_{obj}$, such that, $o^{v}_{obj} = \lambda_1 \lambda_2, ..., \lambda_m (o^{c1}_{obj}, o^{c2}_{obj}, .. o^{ci}_{obj}, .. o^{cn}_{obj})$ where $\lambda_1 \lambda_2, ..., \lambda_m \in \tilde{\lambda}$. That is, o^{v}_{obj} is a newly derived object from existing objects $o^{c1}_{obj}, o^{c2}_{obj}, ... o^{cn}_{obj}$ in the context via a series of conceptual operators $\lambda_1 \lambda_2, ..., \lambda_m$ like select, join, etc.

(ii) For any constraint $\forall c^{v}_{onstraint} \in V^{c}_{constraints}$, there exists a constraint $\exists c^{v'}_{onstraint} \in \zeta_{constraint}$ or a new constraint $c^{v''}_{onstraint}$ constraints associated with V^{c}_{obj} and V^{c}_{rel}.

(iii) For any hierarchical relationship $\forall rel_h \in V^{c}_{rel}$, there *does not exist* a relationship between one or more V^{c}_{obj} and ζ_{obj}.

(iv) For any association relationship/dependency relationships $\forall rel_a \in V^{c}_{rel}$, there *may exist* a relationship between one or more V^{c}_{obj} and ζ_{obj}.

Now we proceed to describe the second level of abstraction, the logical level of the LVM.

5.2.3 Logical (or Schema) Level

The middle level (Fig. 5.1) of the LVM describes the *imaginary document type* of the imaginary XML documents that is the view schema. In this research, we use the XML Schema (W3C-XSD 2001) language to represent the *imaginary document type*. Here, conceptual views (i.e. imaginary classes) defined at the conceptual level are mapped into logical (or schema) views (i.e. imaginary document types) by using the transformation mechanism developed in the works such as; (i) UML to XML Schema (Feng, Chang & Dillon 2003; Xiaou et al. 2001a; Xiaou et al. 2001b) and (ii) XSemantic nets to XML Schema (Feng, Chang & Dillon 2002) (also *see* Chapter 6). The output of this level will be in either textual (such as XML Schema) or some visual notations that comply with the schema language (such as graph).

Definition 5.4. A schema view V^s_{iew} is a 4-tuple with V^s_{name}, $V^s_{simpelType}$, $V^s_{complexType}$ and $V^s_{constraints}$, where, V^s_{name} is the name of the XML schema view V^s_{iew}, $V^s_{simpelType}$, $V^s_{complexType}$ are simple and complex type definitions for XML elements/attributes, and $V^s_{constraints}$ is a set of constraints defined upon the XML elements/attributes.

$$V^s_{iew} = (V^s_{name}, V^s_{simpelType}, V^s_{complexType}, V^s_{constraints}) \qquad (5.7)$$

Here, $V^s_{simpelType}$, $V^s_{complexType}$ and $V^s_{constraints}$ are expressed in the XML Schema Language, and V^s_{name} is also the name of the resulting XML schema file, i.e., a valid W3C XML document name.

In Chapter 3, we presented a transformation function (equation 3.23). Formally, let $\aleph_s^c$ denote the transformation mechanism which can translate a set of objects, their relationships and constraints into a set of simpleType/complextType definitions for XML elements/attributes and associated element/attribute constraints in the XML Schema language.

Definition 5.5. A schema view V_{iew}^s is said to be ***valid***, if and only if, it *satisfies* and *conforms* to the corresponding conceptual view V_{iew}^c definition, from which the schema view V_{iew}^s can be generated using a transformation function $\aleph_c^s$.

Definition 5.6. Given a conceptual view V_{iew}^c, a schema view V_{iew}^s is said to be *semantically* valid schema view, if and only if V_{iew}^s is transformed from V_{iew}^c to V_{iew}^s by a transformation function $\aleph_c^s$.

Let v be conceptual view specification of a conceptual view V_{iew}^c. Thus, from equation 3.23, we can write this as,

$$\aleph_c^s(v) : V_{iew}^c \rightarrow V_{iew}^s \qquad \textbf{(5.8)}$$

In this research, we consider two such transformation functions each of which corresponds to the formalism in which the original conceptual view was defined, namely:

(i) $\aleph_{XSemanticNets}^{XSD}$, to transform conceptual views specified in XSemantic nets to XML Schema, by extending the schemata transformations (to support views) presented in the work (Feng, Chang & Dillon 2002). This is described in detail in Chapter 6.

(ii) $\aleph_{UML}^{XSD}$, to transform conceptual views specified in UML/OCL to XML Schema, by extending the schemata transformations (to support views) presented in the work (Feng, Chang & Dillon 2003; Xiaou et al. 2001a; Xiaou et al. 2001b). This is described in detail in Chapter 7.

The transformation of static properties (conceptual views) in the LVM to logical views (Fig. 5.2) is a one-step transformation (as shown in equation 5.8) that is dependent on the original OO language in which the conceptual view is defined. For example, if the conceptual views are defined using XSemantic nets, the transformation function ($\aleph_c^s$) is $\aleph_{XSemanticNets}^{XSD}$, while if it is UML/OCL, the transformation function is $\aleph_{UML}^{XSD}$. Here, the main point to note is the resulting logical view is always specified using W3C XML Schema language, independent of the original conceptual view specification language, as shown Fig. 5.2.

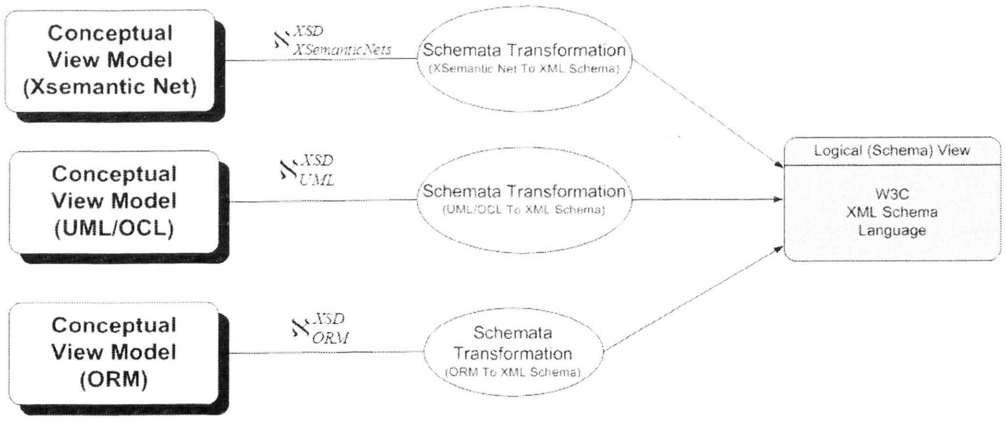

Conceptual View Model (Xsemantic Net)
Conceptual View Model (UML/OCL)
Conceptual View Model (ORM)
$\aleph_{XSemanticNets}^{XSD}$
$\aleph_{UML}^{XSD}$
$\aleph_{ORM}^{XSD}$
Schemata Transformation (XSemantic Net To XML Schema)
Schemata Transformation (UML/OCL To XML Schema)
Schemata Transformation (ORM To XML Schema)
Logical (Schema) View
W3C XML Schema Language

Figure 5.2: Transformation of conceptual views to logical views

It should be noted that, since v_{iew}^s is an XML Schema document, as stated earlier in Chapter 3, equations 3.40, we can also show:

$$v_{iew}^s = \psi_{XSD}^{v_{iew}^s} \qquad \textbf{(5.9)}$$

5.2.4 Document (or Instance) Level

In Fig. 5.1, the bottom instance level designates an instantiated XML data (imaginary instance document), which conforms to the corresponding view schema (logical view) defined at the upper level. Here, the conceptual operators are mapped to query expressions (e.g. XQuery (W3C-XQuery 2004)), which are syntax specific, as explained in Chapter 6.

Let V_{iew}^{d} and V_{iew}^{s} be document and schema views respectively of the corresponding conceptual view V_{iew}^{c}. Formally, we can have the following definition for document views.

As discussed in Chapter 4, during run-time (instantiation), the document view is an imaginary instance document (view instance). It is an XML document.

Definition 5.7. At run-time, an *imaginary instance document* is said to be a ***valid*** and ***well-formed*** document view instance, if and only if, it conforms to the corresponding *valid* schema view V_{iew}^{s} and instantiated by the corresponding *valid* document view V_{iew}^{d}, where the V_{iew}^{d} is the conceptual operator λ that is transformed to λ_{query} by a transformation function $\aleph_{C_{operator}}^{query}$.

Definition 5.8. Given a conceptual operator λ, a document view V_{iew}^{d}, the V_{iew}^{d} is said to be *semantically* valid if and only if λ is transformed to λ_{query} by the transformation function $\aleph_{C_{operator}}^{query}$.

The document views can be constructed via a native XML query language (e.g. XQuery (W3C-XQuery 2004), SQL 2003 (ANSI & ISO 2003)), which can be achieved by the transformation of conceptual operators $\tilde{\lambda}$ defined at the conceptual level into a language-specific query fragment $\tilde{\lambda}_{query}$, say FLWOR expressions in XQuery etc. In a related work (*see* Chapter 9), we have shown how conceptual operators can be transformed and mapped to query algorithms in the MOVE (Wouters 2006) system, which extends and applies this LVM to ontology view extraction.

Let λ be conceptual operator ($\lambda \in \tilde{\lambda}$) involved in construction of a conceptual view V^{c}_{iew} from the context ζ_{ontext}. Thus, from equation 3.23, we can write this as,

$$\aleph^{query}_{C_{operator}}(\lambda) : \tilde{\lambda} \rightarrow \tilde{\lambda}_{query} \qquad (5.10)$$

where, $\tilde{\lambda}_{query}$ is a query expression, such that (from equation 3.50);

$$\tilde{\lambda}_{query} \in Q_{query} \qquad (5.11)$$

In this research, we consider one such transformation function that transforms conceptual operators to XQuery (W3C-XQuery 2004) expression. This can be written as;

$$\aleph^{XQuery}_{C_{operator}}(\lambda) : \tilde{\lambda} \rightarrow \tilde{\lambda}_{XQuery} \qquad (5.12)$$

The formal semantics of XQuery can be found at (W3C-XqFM 2005). It should be noted here that, in the case of transformation of conceptual operators in the LVM to document views, the transformation function is dependent on the document view query language and/or platform (e.g. XQuery, SQL 2003, MOVE system, etc.), as shown in Fig. 5.3. The transformation function $\aleph^{query}_{C_{operator}}$ is equal to $\aleph^{XQuery}_{C_{operator}}(\lambda):\bar{\lambda} \rightarrow \bar{\lambda}_{XQuery}$, if the case of document view query language is XQuery. Also, depending on the target query language, the transformation steps may vary. For example, in the XQuery, the transformation step is one-step, while in the case of MOVE system, the transformation is two-step, as first the conceptual operators are mapped to restricted ontology operators and then the MOVE algorithms.

Thus, the relationship between conceptual operators and the document view query expression is analogous of a conceptual schema for the (document view) query expression. Thus, this research is one of the early such work that proposes a conceptual schema like representation for (executable) query expressions. This is sown in Fig. 5.3.

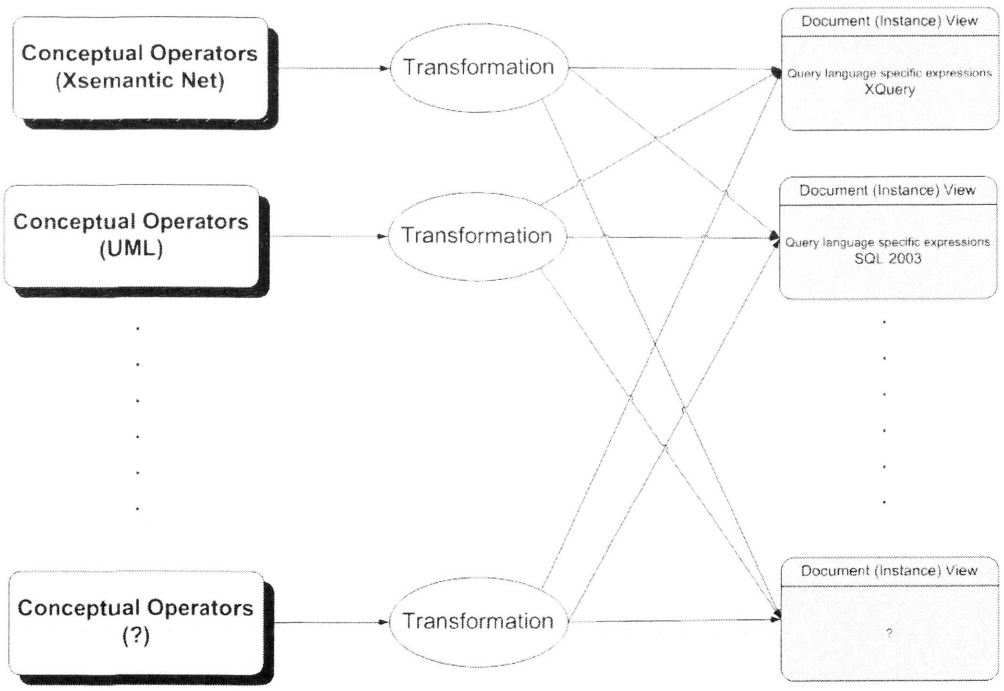


Figure 5.3: Transformation of conceptual operators to document views

Definition 5.9. Given a schema view V^{s}_{iew} and a set of XML source documents ψ, V^{d}_{iew} is called a *valid* document (or instance) view of V^{s}_{iew} over ψ if and only if V_{iew}^{d} is a well-formed XML document, extracted from ψ by certain query operators in λ_{query}, and conforming to the XML schema V_{iew}^{s} .

Let λ_{xq} be an XQuery expression ($\lambda_{xq} \in \lambda_{XQuery}$), $\psi_{XML}^{src} = \{\psi_{XML}^{1}, \psi_{XML}^{2}, ...\psi_{XML}^{j}, ...\psi_{XML}^{m}\}$ be a set of XML source (stored) XML documents, $v_{iew}^{s} (\in V_{iew}^{s})$ be an XML view Schema and $v_{iew}^{d} (\in V_{iew}^{d})$ be an XML document view. Therefore, equation 5.12 may be shown in the context of XQuery as;

$$\lambda_{XQuery}(\psi_{XML}^{src}) \xrightarrow{v_{iew}^{s}} = v_{iew}^{d} \qquad (5.13)$$

It should be noted that, as stated earlier, as v_{iew}^{d} is again a valid XML document. Thus, from equations 3.39 we can also show:

$$v_{iew}^{d} = \psi_{XML}^{v_{iew}^{d}} \qquad (5.14)$$

An imaginary instance document is an instantiated XML document fragment which conforms to the XML schema view defined at the schema level. An instance view can be used for many purposes such as database views, materialised semantic Web-views, etc., and can be generated from simple projection of selected document tags to provide dynamic windows into complex heterogenous documents. Based on how these documents are constructed/behave, as described in Chapter 4, Section 4.9, we classify XML instance views into three categories, namely: *derived imaginary instance documents, constructed imaginary instance documents,* and *triggered imaginary instance documents.*

5.2.5 View in the LVM

In the previous section we presented various layers of abstraction, components, their properties and definitions of the LVM. Here, we look at an XML view definition in the LVM, in totality.

Given the definitions and properties above, we can write the XML view in the LVM (V_{LVM}^{XML}) as a function of V^c, V^s and V^d, where $V^c \in V_{iew}^c, V^s \in V_{iew}^s$ and $V^d \in V_{iew}^d$.

Definition 5.10. A layered XML view V_{LVM}^{XML} is a 3-tuple with V^c, V^s and V^d, where V^c is the *conceptual view* of the layered XML view in V_{LVM}^{XML}, V^s is the *schema (or logical) view* of the layered XML views in V_{LVM}^{XML}, V^d is the *document (or instance) view* of the layered XML view in V_{LVM}^{XML}.

$$V_{LVM}^{XML} = (V^c, V^s, V^d) \qquad (5.15)$$

Here V^c provides the (conceptual) view definition, V^s provides the schema and V^d provides the query expression for the layered view. Let v_{LVM}^{XML} be an XML view in the LVM and, v^c, v^s, v^d be the corresponding conceptual, logical and document views related to the XML view v_{LVM}^{XML}, respectively. Then we can write $v_{LVM}^{XML} = (v^c, v^s, v^d)$.

Definition 5.11. An XML view v_{LVM}^{XML} in the LVM is said to be ***valid*** if and only if, v^c of v_{LVM}^{XML} is a *valid* conceptual view, the schema view v^s of v_{LVM}^{XML} is a valid schema view against the conceptual view v^c and the document view v^d of v_{LVM}^{XML} is a *valid* document view against the schema view v^s.

5.3 Conceptual Operators

As described in Chapter 4, to enable the construction of a valid conceptual view from a context, we introduce the notion of *conceptual operator* $\tilde{\lambda}$. These conceptual level operators are analogous to relational operators in the relational model, but they operate on conceptual level objects and relationships. In the following subsection we present some of the formal semantics of these operators.

In the relational model, relations serve as the descriptive schema for the tables with tuples as rows and attributes as columns. Thus, the relational operators operate on one or more such relations (operand(s)) to obtain a result by vertical and/or horizontal partitioning of the relations. The result is again a relation with columns and attributes with tuples projected, derived or manipulated from the original stored tuples.

Conversely, with the conceptual operators, unlike relational or other operators, the operands are of abstract nature. That is, the input and output for the operators are conceptual objects, such as classes, relationships, constraints, instance objects etc. Therefore, in the case of conceptual operators, context (i.e. a set of classes, their object schemas and their description in the given OO modelling language) is analogous to a relation, while the objects in the context (we refer to them here as items) are analogous to the tuples (i.e. rows and columns in the relational model). In contrast to relational tuples, in the contexts, a class may refer to a simple class with attributes or a class hierarchy, such as composition.

For example, as shown in Fig. 5.4, in the example CPS case study as described in Chapter 4, there are three examples of contexts are shown, namely `paper`, `conference` and `people` using the UML as the modelling language. The context `people` has members such as, `Person`, `Organization`, `Author`, `Referee` and `PC-Member` as its members. Thus, one or more of these classes can be operand(s) and their instance objects be items for the conceptual operators.

An equivalent model of the CPS given in Fig. 5.4 is shown in Fig. 5.5, using the XSemantic net notations as described in Chapter 3. Here, similar to the case of

UML, a set of connection cluster sets will be operands and their of-property relationships and their values (i.e. basic nodes) will be items for the conceptual operators.

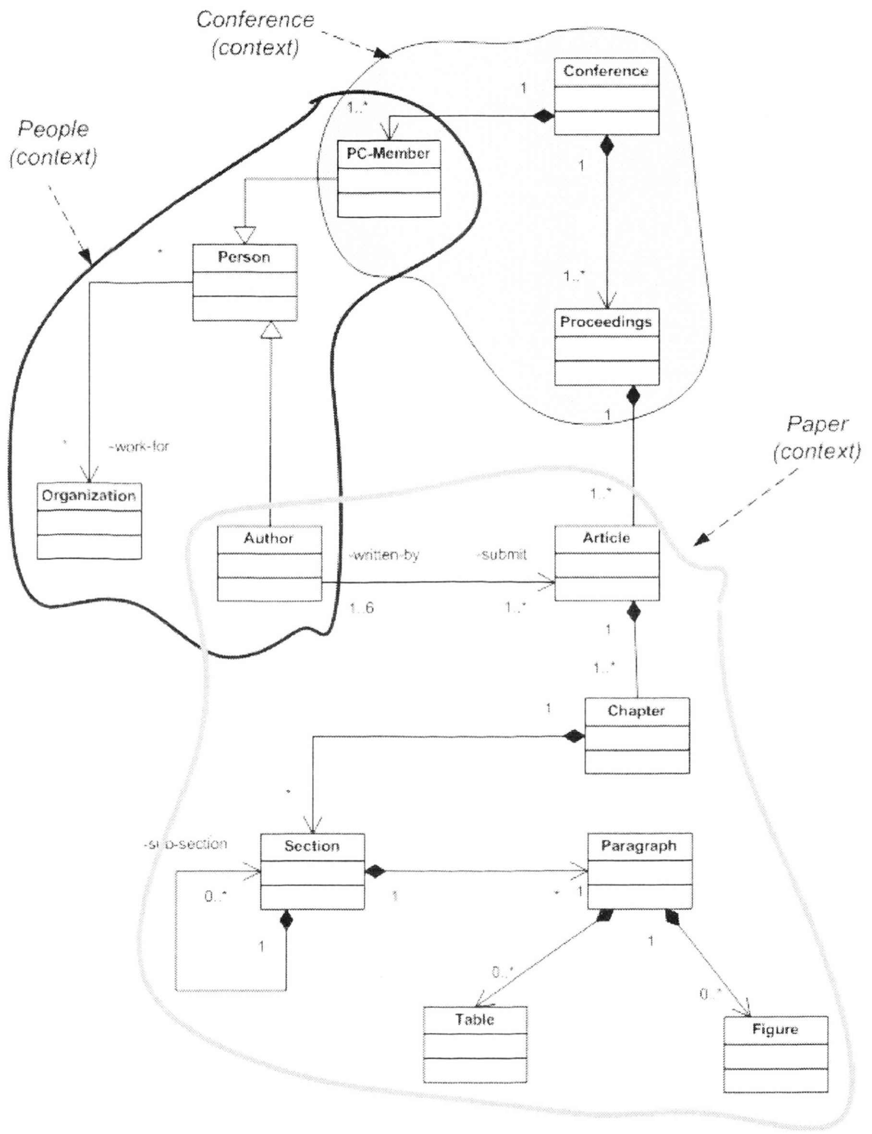


Figure 5.4: CPS and examples of contexts (UML representation)

Example 5.4: In order to group objects for all the authors who are also PC-members, we can apply the intersection operator to the context people with operands PC-Members and Authors respectively.

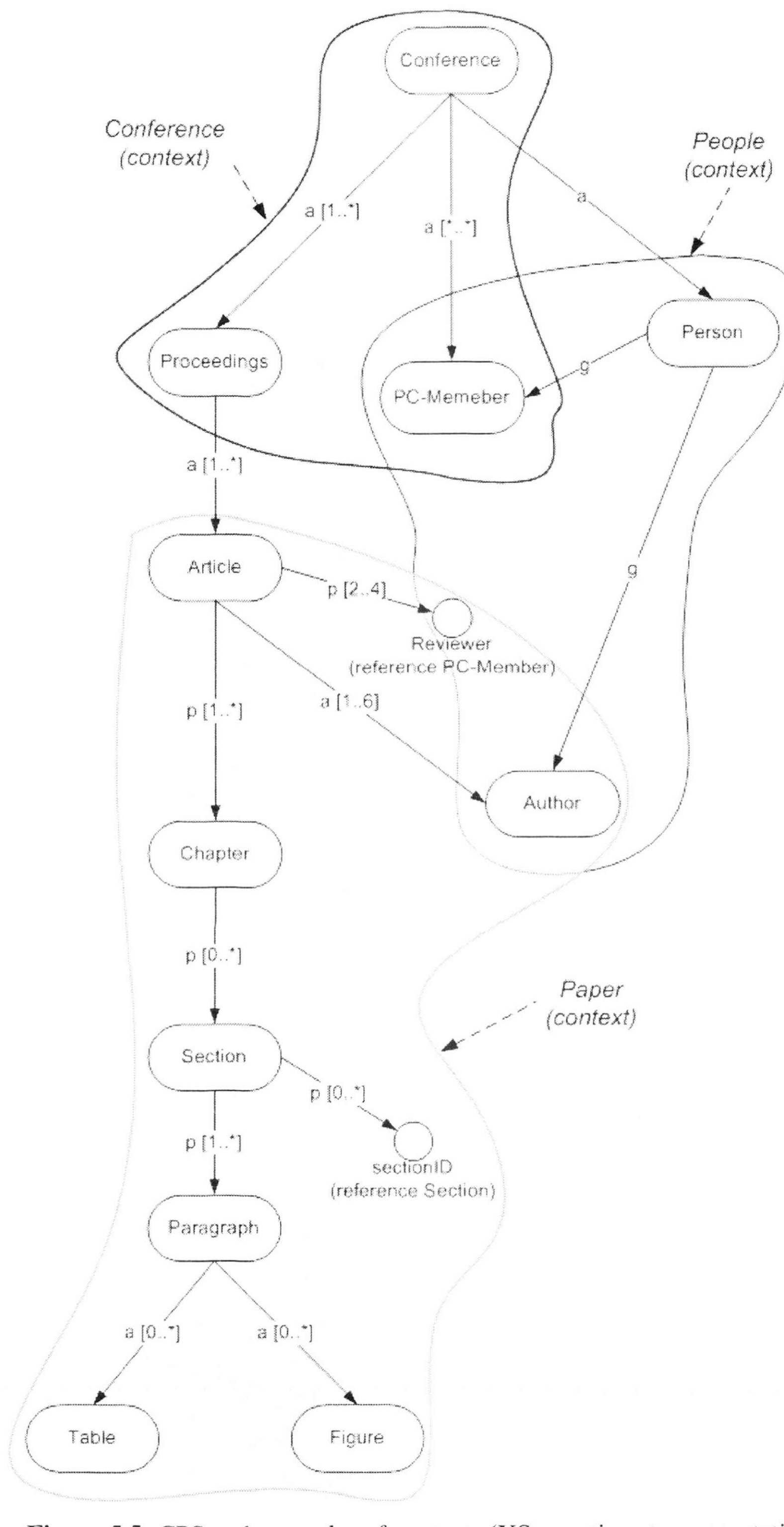


Figure 5.5: CPS and examples of contexts (XSemantic net representation)

Example 5.5: Similarly, to group all PC-members who did NOT submit papers to the conference, we can apply the difference operator to the context people with operands PC-Members and Authors respectively.

Example 5.6: Similarly, to group all PC-members who did NOT submit papers to the conference, we can apply the difference operator to the context people with operands PC-Members and Authors respectively.

Example 5.7: To select reviewers, we can apply a selection criteria to PC-Members whose role= "reviewer". This can be achieved by using the selection operator, in the context of people with PC-Members as the operand.

Example 5.8: To create a list of login details for the authors, one can apply union and projection operator in sequence to get authorID, userName and password, in the context of people with PC-Members, Authors as the operands to union operator, and the result as an operand to the project operator.

Example 5.9: To create a list of abstracts with paper-abstract, paper-details, author-details and paper-keywords, one can apply the join and projection operator in sequence to get the required details in the context of paper.

Let X,Y be two objects in a given context ζ_{ontext}, ($X,Y \in \zeta_{obj}$ are called operand$_1$ and operand$_2$ respectively) that belong to the domains $D(X) = dom(X)$ and $D(Y) = dom(Y)$ respectively. Let R be a valid conceptual view such that $R \in V^{c}_{iew}$ and has the domain of $dom(R)$. In addition, let the instance objects of X,Y and R be denoted ass x, y and r respectively (referred to as "items" here). Also, $p[Z]$ denotes the value of item p on an item set Z such that $Z \subseteq X$ (or $Z \subseteq Y$).

5.3.1 Conceptual Binary Operators

The conceptual binary operators take in two operands produces a result set. The following algebraic operators are defined for constructing conceptual views in the LVM.

5.3.1.1 Union Operator

A *union* operator $\bigcup_{(X,Y)}$ of operands X,Y will produce a result R such that it includes all the items that are either in X or in Y or in both X and Y with *no* duplicates. This can be written as:

$$\bigcup_{(X,Y)} = R = X \cup Y = X \cup X' = Y \cup Y' \qquad \textbf{(5.16)}$$

where, $dom(R) = D(X) \cup D(Y)$

Definition 5.12. If X and Y are two union-compatible operands, and R is the result set, then $\bigcup_{(X,Y)}$ is defined such that, it includes all items either in X or in Y or in both X and Y with duplicates removed, as shown below:

$$\bigcup_{(X,Y)} = \{r \mid r \in x \vee r \in y\} \qquad \textbf{(5.17)}$$

where, $X,Y \in \zeta_{obj}$.

5.3.1.2 Difference Operator

A *difference* operator $\overline{D_{(X,Y)}}$ of operands X and Y will produce a result R, ($R \in V^{c}_{iew}$), such that it includes all items that are in X but not in Y .

$$\overline{D_{(X,Y)}} = R = X - Y \qquad \textbf{(5.18)}$$

where $dom(R) = D(x)$ or $dom(R) \subseteq D(x)$.

Definition 5.13. If X and Y are two union-compatible operands, and R is the result set, then $\overline{D_{(x,y)}}$ is defined such that, it includes all items in X but *not* in Y, as shown below:

$$\overline{D_{(x,y)}} = \{r \mid r \in x \wedge r \notin y\} \qquad \textbf{(5.19)}$$

where, $X, Y \in \zeta_{obj}$.

5.3.1.3 Intersection Operator

An *intersection* operator $\bigcap_{(X,Y)}$ of operands X and Y will produce a result R, such that it includes all items that are in both X and Y .

$$\bigcap_{(X,Y)} = R = X \cap Y \qquad \textbf{(5.20)}$$

where $dom(R) \subseteq D(x)$ or $dom(R) = D(x)$.

Definition 5.14. If X and Y are two union-compatible operands, and R is the result set, then $\bigcap_{(X,Y)}$ is defined such that, it includes all items that are in both X and Y , as shown below:

$$\bigcap_{(x,y)} = \{r \mid r \in x \wedge r \in y\} \qquad \textbf{(5.21)}$$

where, $X, Y \in \zeta_{omext}$.

It should be noted that both union and intersection operators are *commutative* and *associative* and they can be applied to n-ary operands, that is, for $\bigcup_{(X_i,X_j)}$, where $1 \le i \le n; 1 \le j \le i$:

$$\bigcup_{(X_i,X_j)} = X_1 \cup X_2 \cup ....X_i \cup X_j ... = X_{n-1} \cup X_{n-2} \cup ...X_j \cup X_i .... \quad (5.22)$$

and

$$\begin{aligned} \bigcup_{(X_i,X_j)} &= (X_1 \cup X_2) \cup X_3 ...X_i \cup X_j ... \\ &= X_1 \cup (X_2 \cup X_3) ...X_i \cup X_j .... \end{aligned} \quad (5.23)$$

Similarity for $\bigcap_{(X_i,X_j)}$ where $1 \le i \le n; 1 \le j \le i$:

$$\bigcap_{(X_i,X_j)} = X_1 \cap X_2 \cap ....X_i \cap X_j ... = X_{n-1} \cap X_{n-2} \cap ...X_j \cap X_i .... \quad (5.24)$$

and

$$\begin{aligned} \bigcap_{(X_i,X_j)} &= (X_1 \cap X_2) \cap X_3 ...X_i \cap X_j ... \\ &= X_1 \cap (X_2 \cap X_3) ...X_i \cap X_j .... \end{aligned} \quad (5.25)$$

It should be noted that the difference operator is NOT *commutative*.

5.3.1.4 Cartesian Product Operator

A *Cartesian product* operator $\times_{(X,Y)}$ of operands X and Y will produce a result R, such that it includes all items of X and Y combined in combinatorial fashion.

$$\times_{(X,Y)} = R = X \times Y \qquad (5.26)$$

where $dom(R) = D(x) \cup D(y)$

Definition 5.15. If X and Y are two union-compatible operands, and R is the result set, then $\times_{(X,Y)}$ is defined such that, it is a context $X \cup Y$ with a combination of a an item from X and an item from Y, as shown below:

$$\times_{(x,y)} = \{r(x \cup y) \mid r[X] \in x \wedge r[Y] \in y\} \qquad (5.27)$$

where, $X, Y \in \zeta_{ontext}$ and the domain .

Let a, b and N be the number of items in X, Y and $\times_{(X,Y)}$ respectively. Thus the total number of items in $\times_{(X,Y)}$, denote by $|\times_{(x,y)}|$ is equal to:

$$|\times_{(x,y)}| = N = a * b \qquad (5.28)$$

5.3.1.5 Join Operator

Given two operands, a *join* operator can be shown in its general form as:

$$\rhd\lhd_{(X,Y)} = R = X \rightarrow_{j_{condition}} Y \qquad (5.29)$$

where, $X, Y \in \zeta_{obj}$ and an optional join-condition provides meaningful merger of objects in a given context. The join-condition $j_{condition}$ is a *conditional predicate* in the form of $A\,\theta\,B$, where A, B are two context such that, $A \in X$, $B \in Y$

and $\theta \in \{>, <, \geq, \leq, =, \neq\}$. Also, depending on the content type of the join-condition may be of:

(i) simple-condition: where the join-condition $j_{condition}$ is specified using $X, Y \in \zeta_{obj}$ that are of simple content $s_{content}$ types,

(ii) complex-condition: where the join-condition $j_{condition}$ is specified using sing $X, Y \in \zeta_{obj}$ that are of complex content $c_{content}$ types and

(iii) Pattern-condition: where the join-condition $j_{condition}$ is specified using a combination of mixed content, i.e. $X, Y \in \zeta_{obj}$ are a mix of simple and complex content types in a hierarchy with additional properties and constraints.

Definition 5.16. If X and Y are two union-compatible operands, and R is the result set, then the items in $\rhd\lhd_{(X,Y)}$ is defined as the combination of a related items from X and a related item from Y that satisfies respectively the given condition in $j_{condition}$. This is shown below:

$$\rhd\lhd_{(X,Y)} = \{r(x \cup y) \mid r[X] \in x \wedge r[Y] \in y \wedge j_{condition}(r[X], r(Y))\} \qquad \textbf{(5.30)}$$

A join operator may be also stated using select and a Craterisation product operator, as described in section 5.3.2.2 below.

5.3.1.5.1 Natural Join

A natural join operator$\rhd\lhd_{(X,Y)}$ of operands X, Y is a join operator with no join-condition, that is, in the join-condition A, B are identical and θ takes "=", will produce a result R, such that R is equivalent to a Cartesian product operator $\times_{(X,Y)}$, as shown below.

$$\rhd\lhd_{(X,Y)} = R = X \rhd\lhd Y = \times_{(X,Y)} \qquad \textbf{(5.31)}$$

This can be also shown as:

$$\begin{aligned}&\rhd\lhd_{(X,Y)} = \\ &\{r(x \cup (y - (x \cap y)) \mid (\exists a)(\exists b)(a \in x \wedge b \in y \wedge a[X \cap Y] \\ &= b[X \cap Y] \wedge r[X] = a[X] \wedge r[Y - (X \cap Y)] \\ &= b[Y - (X \cap Y)]\}\end{aligned} \qquad \textbf{(5.32)}$$

5.3.1.5.2 Conditional Join

A join operator $\rhd\lhd_{(X,Y)}$ of operands X,Y with explicit join-condition $j_{condition}$ specified will produce a result R, such that R will have *only* the combination that satisfies the join-condition $j_{condition}$.

$$\rhd\lhd_{(X,Y)} = R = X \rightarrow_{(j_{condition1}[AND.....])} Y \qquad \textbf{(5.33)}$$

5.3.1.5.3 Pattern Join

A join by pattern $\rhd\lhd_{(X,Y)}$ is a join by condition operator where the join-condition $j_{condition}$ is specified using a mixed content of simple and/or complex contents.

5.3.2 Conceptual Unary Operators

We propose four unary conceptual operators to construct conceptual views without loss of CO semantic that are represented in the model. The four conceptual operators are projection, selection, rename, and restruct(ure).

5.3.2.1 Projection Operator

Given a valid context X, the projection operator $\prod_z(X)$ will produce a result R, where it has only the specified items specified in the item set Z (where, $Z \subset X$) with: (a) persevered node hierarchy, (b) preserved node order and (c) preserved semantic relationships (if any). If need to , item set Z (in the case of complex content) may be specified using the W3C XPath (W3C-XPath 1999) standard.

$$\prod_Z(X) = R = \prod_{(z_1, z_2, \ldots)}(X) \qquad (5.34)$$

where the domain of R is $dom(R) = \bigcup_{k=1}^{m} dom(z_k) = dom(Z)$

Definition 5.17. Given a valid context X and an item set Z, the project operator $\prod_z(x)$ produce result set R such that, it includes all the items that are given in Z such that, $Z \subset X$. This is shown below:

$$\prod_z(X) = \{r(Z) \mid (\forall a)(a \in x(Z) \wedge t = a[Z])\} \qquad (5.35)$$

5.3.2.2 Selection Operator

Given a valid context X, the select operator $\sigma(X)$ will produce a result R, where it contains one or more matching items (or collections) that satisfy the Boolean select-condition $s_{condition}$. In addition, the select-conditions may be simple or composite with the combination of logical operators AND, OR, NOT logical operators.

$$\sigma(X) = R = \sigma_{s_{condition}}(X) \qquad (5.36)$$

Similar to the join-condition, here, the select-condition $s_{condition}$ be of the form:

(i) simple-condition: where the select-condition $s_{condition}$ is specified using simple content $s_{content}$ types and the select operator is called value-based;

(ii) complex-condition: where the select-condition $s_{condition}$ is specified using complex content $c_{content}$ types and the select operator is called structure-based; and,

(iii) pattern-condition: where the select-condition $s_{condition}$ is specified using a mix of simple and/or complex content types in a hierarchy with additional constraints, such as ordering etc, where the select operator is called pattern-based.

Definition 5.18. Given a valid context X and a Boolean predicate expression (select-condition), the selection operator $\sigma(X)$ will produce a result set R such that, it will only include items in X that satisfy the given selection condition $s_{condition}$, as shown below:

$$\sigma_{s_{condition}}(X) = \{r \mid r \in x \wedge s_{condition}(r)\} \qquad (5.37)$$

5.3.2.3 Rename Operator

Given a valid context X and a src (with old and new labels), the rename operator $\rho_{src}(X)$ will return R where the label of X is changed.

$$\rho_{src}(X) = R = \rho_{l^{old},l^{new}}(X) \qquad (5.38)$$

where src is a pair such that $src = (l^{old}, l^{new})$ and each member of src is $l^{old}, l^{new} \in \zeta_{name}$.

A RENAME operation cannot; (a) alter X specific data types and (b) alter src specific contents, values or constraints.

5.3.2.4 Restructure Operator

Given a valid context X and a src (with a pair of positions, old and new (pos_1, pos_2)), where the positions can be either absolute or relative (in the context hierarchy), the restructure operator $\delta_{src}(X)$ will return R, where the position of X (can be either $s_{content}$ or $c_{content}$) is changed from pos_1 to pos_2.

$$\delta_{src}(X) = R = \delta_{(pos_1, pos_2)}(X) \qquad (5.39)$$

But a restructure operation does not allow: (a) deletion of context in the hierarchy; (b) alter structural relationships, constraints, names or cardinality; and (c) alter data types or values.

5.3.3 Derived Operators

The operators presented above (except join and intersection) are referred to as the primitive or non-restive, basic set, as many other secondary operators (e.g. division, intersection, restrictive operators etc.) can be derived by combining one or more of these binary and unary operators. In this section, we describe how other operators can be derived from the given primitive set of operators (i.e. union, difference, Cartesian product, projection, selection, rename and restructure operators). Some of the operators described above (e.g. natural join) may also be described using this primitive set. The following sections discuss some of these operators in detail. Chapter 9 also preset some such non-primitive operators, in the context of XML Document Warehouse (XDW) and ontology views, respectively.

5.3.3.1 Division Operator

Division operators are used to extract sub-components of an operand containing all the sub items. A division operator $\div_{(X,Y)}$ of operands X, Y will produce a result R, such that it includes all sub-items of X which have for complements in X, all the items of Y. This may be shown as:

$$\div_{(X,Y)} = R = X \div Y \qquad (5.40)$$

where $dom(R) = D(x) \cup D(y)$

Definition 5.19. If a is an item of Y and x is a sub-item of X, a division operator is defined such that, an item (x,a) is an item of X, as shown below:

$$\div_{(X,Y)} = \{r \mid (\forall a)(a \in y \wedge (r,a) \in x\} \qquad (5.41)$$

where $X, Y \in \zeta_{obj}$.

Alternatively, the division operator may be also stated using project, difference and Cartesian product operators, such that:

$$\div_{(X,Y)} = \prod_{(X-Y)}(X) - \prod_{(X-Y)}(x(X-Y) \times Y - X) \qquad (5.42)$$

5.3.3.2 Compression Operator

A compression operator performs compression of a given set of items (of the context) and indicates that those item sets are replaced by a single item set in the result. The item set itself can be a new item, but it will not provide additional semantic information (compared to the original context). The compression operator is a complex operator, constituted of the concatenation of multiple unary operations applied in sequence.

A detailed formal definition of the compression operator is given in (Wouters 2006), in the context of ontology views and discussed briefly in Chapter 9.

5.3.3.3 Join Operator

As describe before, join operators help one to connect and query semantically related contexts. It can also be stated using selection and Cartesian product operators, as:

Definition 5.20. If X and Y are two union-compatible operands, and R is the result set, then the items in $X \rhd\lhd_{j_{condition}} Y$ is equivalent to selection operator $\sigma_{s_{condition}}(R)$, where R the result of a Cartesian product operator $\times_{(X,Y)}$ and the select condition $s_{condition}$ equal to the join-condition $j_{condition}$. This may be shown as:

$$\rhd\lhd_{(X,Y)} = X \rhd\lhd_{j_{condition}} Y = \sigma_{s_{condition}}(X \times Y) \qquad (5.43)$$

where $s_{condition} = j_{condition}$.

5.3.3.4 Intersection Operator

As described before, intersection operators may be stated in terms of a difference operator as given in definition 5.12 and in equation 5.23. Thus, we can write an intersection operator as:

$$\cap_{(X,Y)} = X - (X - Y) \qquad (5.44)$$

5.3.4 Some Examples of the Conceptual Operators

Example 5.10: In the CPS example, `Conference-Chairs`, which lists the entire conference chairpersons in the system for this year, is a valid conceptual view, constructed in the context of `Committee-Members`. It is constructed using the selection operator, which can be shown as:

$$\sigma_{memberRole="Chair" \, AND \, memberYear=2006}(Committe - Memebers) \qquad (5.45)$$

Example 5.11: Similarly, in the CPS, Special-Review-Paper is a conceptual view, in the context of Paper, if the authors are part of the organizing committee of the conferences. This view can be constructed as:

$$\sigma_{Author/PCMember=true}(Paper) \qquad (5.46)$$

Example 5.12: If reviewers need to be notified of review report due dates, a message can be attached to their profiles using Cartesian Product operator as: let et `x` = `Reviewer/chairMsg` and `y` = `Conference-Chairs/msgToPCMember`:

$$\times_{(x,y)} = x \times y \qquad (5.47)$$

Example 5.13: In the CPS, `Address-Book` is a valid conceptual view in the context of `person`, which can be shown with the projection operator as:

$$Address - Book = \prod_{streetNo, streetName, suburb, city, postcode, country} (Person) \qquad (5.48)$$

Example 5.14: Another valid conceptual view `Author-Notification` may be generated based on `Reviewer-Reports`, `Paper` and `Author` records, in the context of Notifications; let `X` = `Reviewer-Reports`, `Y` = `Paper` and `Z` = `Author`:

$$\begin{aligned} &Author - Notification_{DEXA2006} = \\ &\rhd\lhd_{(X,Y,Z)} = (X \rightarrow_{Reviwer-Report.paperID=Paper.paperID} Y) \\ &\qquad AND\,(Y \rightarrow_{(Paper.authorID=Author.paperID)} Z) \end{aligned} \qquad (5.49)$$

5.4 Modeling Conceptual Views

The conceptual views are views that are captured at the conceptual level with conceptual semantics and constraints. To the best of our knowledge, there exists no notation to specify or define such high level views or view semantics using existing OO modelling languages. Even in the ORA-SS view model (Chen, Ling & Lee 2002), the views are only represented (in contrast to defining) using a diagrammatic notation. Thus, in the layered view model, to define and specify views at the conceptual level (*conceptual views*), a new set of notations needed to be developed. These include: (i) a notation to represent conceptual views, visually, at the conceptual level; (ii) a set of notations to represent conceptual view properties; (iii) a notation to represent conceptual view constraints; and (iv) a notation to represent view constructs.

As our conceptual view mechanism is defined at a higher-level of abstraction, we can provide an explicit view constraint specification model, as most high-level OOCM languages (such as UML, XSemantic nets, E-ER) provide some form constraint specification. In the case of XSemantic nets, constraints are provided as part of the model elements (Feng, Chang & Dillon 2002).

The discussion on the modelling and transformation of conceptual views are discussed in Chapters 6 and 7. In the following sections, some of the theoretical concepts related to conceptual views and view properties are discussed.

5.5 Constraint Specification in Conceptual Views

As we stated in Chapters 3 and 4, conceptual views are represented and/or a equivalent to the notion of a class in OO modelling languages. A (conceptual) view class is a class with attributes, and optional methods and constraints associated with its attributes and methods.

Also, as we stated in Chapter 4, constraint specification for conceptual views can be explicit and detailed, specified at the conceptual level. Also, the constraint specifications are restricted only by the modelling language semantics and not by the view definition language as in the case of relational (e.g. SQL) and/or OO (e.g. OQL) views. Here, we look at providing descriptive semantics for conceptual view constraints. The conceptual views constraints include;

(i) unique constraints

(ii) referential constraints

(iii) ordered composition

(iv) explicit homogenous composition/heterogeneous compositions,

(v) adhesion and/or dependencies

(vi) exclusive disjunction

(vii) cardinality constraints

(viii) domain constraints (range of values, min, max, pattern etc)

(ix) constructional contents (set, sequence, bag, ordered-set)

Therefore, conceptual view constraints $V^c_{constraints}$ may be described as a collection of various types of constraints and shown as a set, such that:

$V^c_{constraints}$ = { `unique (), reference (), order (),`

```
homogenous (), exclusive (),
{<adhesion>},
{<domain-constraints>},
{<cardinality-constraint>},
constructional () }
```

(i) Unique Constraint

In an OO system, an Object has a unique system-wide identifier that is independent of the values of its attribute/(s), called Object Identifier or OID (Dillon & Tan 1993; Doorn, Rivero & (eds) 2002). When created, an object will be referred to using its system assigned OID during its lifetime. In DBMS systems, OIDs can be either *logical* or *physical* depending on it nature. In a DBMS environment, physical OID may contain physical object address while a logical OID points to a logical ID which may get *mapped* to the object's physical object address using some algorithm (B+ tree, hash table etc).

For example, a unique constraint for an attribute may be specified as,

$V^{c}_{constraints}$= `unique (<attribute-name>)`

In many OO conceptual models and diagrams, though the concept of OID is assumed to be an implicit concept (unlike primary keys in E/ER), in conceptual views, there is a need to *explicitly* state OIDs, and is available to be visualized at the conceptual level. This is mainly due to the nature of key-independent XML trees. Here, OIDs provide a means of using them for the purpose of IDs, similar to that of primary/foreign key constraints available in the ER models. We argue that, just utilizing OID (a unique concept to OO systems) in our conceptual model provides additional semantics such as providing Id/keys, referential and integrity constraints that are visually lacking in many OO conceptual modelling technique.

Let a_i, a_j be two instances of an attribute A_X $(a_i, a_j \in A_X)$ that belonging to conceptual view $V_X \in V^c$, where $0 < i, j \leq n$ and $0 < i, j \leq n$ is the total number of instances of the conceptual view V_X .

Definition 5.21. An attribute A_X of a conceptual view V_X is said to be unique, if and only if the following condition holds true:

$$unique(A_X) = (a_i, a_j \in A_X \mid a_i \neq a_j \wedge i \neq j) \qquad \textbf{(5.50)}$$

Example 5.15: In the CPS, `authorID`, `paperID` are unique, which can be described as:

$V^c_{constraints}$ = `unique (paperID)`

$V^c_{constraints}$ = `unique (authorID)`

(ii) Referential Constraint

The referential constraint requires that, in order to have a valid referential constraint, there needs to exist a (view) class or (view) attribute content that is equal or the same in value to another (view) class or (view) attribute. Referential constraints help to maintain referential integrity of a domain.

$V^c_{constraints}$ = `<A> reference <B>`

- `A` = V^c_1 , `B` = V^c_2, where is (V^c_1, $V^c_1 \in$ conceptual view V^c) and $V^c_1 \neq V^c_1$
- `A` = `view-attribute`$_1$, `B`= `view-attribute`$_2$

(iii) Ordered Composition

In a real-world scenario, in an aggregation relationship (i.e. in conceptual view hierarchy), there may exist a "whole" object that is made of "part" or composite objects that occur in a predefined order. This signifies an important relationship, *ordered composition*. For example, in XML Schema construct such as with `<xsd:sequence>`, we regularly observe that the tag `<xsd:sequence>` signifies that the embedded elements are not only a simple assortment of components but these have a specific ordering. That is, a whole object (e.g. conceptual view) may have its sub-components in a given order.

For example, ordering of a finite set of "part-of "conceptual views V^c_r, ($V^c_r \in V^c$) can be shown as,

$V^c_{constraints}$ = `order` (V^c_1, V^c_2, ... V^c_n)

Example 5.16: The composition relationship between `Paper` (whole) and `Abstract` (part), `Section` (part) is *ordered*. This can be described as:

$V^c_{constraints}$ = `order (Abstract, Section)`

(iv) Homogenous Composition

In a real-world scenario, in an aggregation relationship (i.e. in conceptual view hierarchy), there may exist a "whole" object that is made up of "part" or composite objects that are of the same type as the "whole" object. This kind of whole-part relationship is referred to as *homogenous* composition. From a modelling point of view, explicitly stating homogenous compositions in modelling conceptual views is beneficial in designing better schema or programming structures. For example, homogenous composition of a finite set of "part-of" conceptual views V^c_r, ($V^c_r \in V^c$) is described as,

$V^c_{constraints}$ = `homogenous` (V^c_1, V^c_{11}, V^c_{12}, ... V^c_{111}, ...)

Example 5.17: For example, in CPS there exists homogenous composition between `Section` and `Sub-Section`, which can be described as:

$V^c_{constraints}$ = `homogenous(paper-sections, paper-sub-sections)`

(v) Exclusive disjunction

Exclusive disjunction constraint may be applied to both aggregation and association type relationships between conceptual views. It implies that only one conceptual view class exclusively connected in a relationship. For example, exclusive disjunction of a finite set of conceptual views V^c_r, ($V^c_r \in V^c$) is described as:

$V^c_{constraints}$ = `exclusive` (V^c_1, V^c_2, ... V^c_n)

Example 5.18: For example, in CPS, mailing of proceedings has an exclusive disjunction between `shipping-AirMail` and `shipping-SeaMail` is shown below:

$V^c_{constraints}$ = `exclusive (shipping-AirMail, shipping-SeaMail)`

Example 5.19: Another example in CPS, where author cannot review his own paper may be stated as an an exclusive disjunction constraint, which can be shown as:

$V^c_{constraints}$ = `exclusive (author, referee)`

(vi) Cardinality Constraint

The cardinality constraint shows the number of instances of one view class that may relate to single instance of another. This is similar to that of relational and OO data models. Cardinality constraints include one-to-one (1..1), one-to-many (1..*), zero-or-one (0..1), zero-or-more (0..*), and many-to-many (*..*). This can be shown as:

$V^c_{constraints}$ = `<cardinality-constraint> ::=`

```
[1..1] | [1..N] | [0..1] | [0..N] | [N..M]
```

(vii) Dependency / Adhesion Constraint

Adhesion (or dependency) constraint indicates whether participants of a relationship (aggregation, association, dependency, instance etc.) should coexist and hold fast to each other. Also, in the case of class-attribute relationships, this constraint may be used to indicate optional or compulsory attributes, while in most semantic relationships (aggregation, association) the adhesion relationships is always of type strong adhesion. Adhesion relationship is described as:

$V^{c}_{constraints}$ = `<adhesion> ::= [strong-adhesion] | [weak-adhesion]`

This is also the case in conceptual views, and the class-view relationships (i.e. dependency or *construct,*) the relationship is always *strong adhesion*. It should be noted that, class-view is one special type of the strong adhesion/dependency relationship.

Example 5.20: For example, in CPS, the relationship between `paperID` and `AuthorID` is a strong adhesion, as a `paperID` cannot exist without `authorID`. This can be described as:

$V^{c}_{constraints}$ = `adhesion (paperID, authorID)= strong-adhesion`

Example 5.21: In the e-Sol, there exists a storing adhesion between `Income` and `Salary` and `Benefit-Packages` which can be described as:

$V^{c}_{constraints}$ = `adhesion (Income, Salary-Package)`

```
                = strong-adhesion
                AND
          adhesion (Income, Benefit-Package)
                = strong-adhesion
```

Example 5.22: Another example, adhesion relationship between `Referee` and `Paper` as such entry is optional, i.e. review may not review any papers. This can be described as:

$V^{c}_{constraints}$ = `adhesion (Referee, Paper) = weak-adhesion`

(viii) Domain Constraints

The notion of domain constraints includes a wide range of constraint properties that can be applied to the data values (and types) in a given domain. For example, the domain constraints for conceptual views include; (i) restriction of values, range of values and lengths of data types; (ii) enumerated and/or pre-specified (default) values; (iii) pre-defined patterns (e.g. date formats) etc. These constraints can be shown as:

$V^{c}_{constraints}$ `= {<domain-constraints>}`

```
<domain-constraints> ::=      [Val v] | [MinVal v] |
                         [MaxVal v] | [Len v ] |
                         [MinLen n]|[MaxLen n] |
                         [Val s] | Pattern s]
```

where `v` is a numerical value, and `s` is a string.

Example 5.23: In the CPS,

- Maximum length of the `reviewerID` field is 10,

$V^{c}_{constraints}$ `= reviewerID := (MaxLen = 10)`

- `paperSubmission` date may be expressed in two formats;

$V^{c}_{constraints}$ `= paperSubmisison := (MaxLen = 10),`
`(Pattern dd-mm-yyyy)`

- `Referee` (and `Author`) access code/password should have minium 6 characters

$V^{c}_{constraints}$ `= password := (MinLen = 6)`

(ix) Constructional Constraints

Constructional constraints are constraints associated with constructional data types and values such as set, list, bag and union. Conceptual views can be defined based on one (or more) stored type and constructs such as set, list, bag etc. of a particular type. Here, apart from union type, all members in the constructional value must be of same type.

(i) A *set* value is an unordered collection type without duplicates. Here, all members are unique and of the same type. Also, additional restrictions may be specified as part of the list constructional constraints, namely: (a) total length of the list, (b) minimum length allowed, (c) maximum length allowed, and (d) enumeration.

(ii) A *bag* value is an unordered collection type with duplicates.

(iii) A *list* value is an ordered collection type with duplicates. A list is a sequence of atomic types composed to represent one type. Furthermore, similar to set and bag, additional restrictions may be specified as part of the list constructional constraints, namely: (a) total length of the list, (b) minimum length allowed, (c) maximum length allowed, and (d) enumeration.

(iv) A *union* value is a combination of one or more types combined to provide more than one value for a given type. Unlike set, list or bag, union allows its basic types to be of multiple types. It should be noted here that the union constraint described here is different from the union operator described in Section 5.3.1.1. Though both semantics are analogous, their application and semantics differ in context (here union refer to a data type construct while the union conceptual operator refers to the construction of conceptual views) and application.

$V^{c}_{constraints}$ `::= [set | list | bag | union] (<attribute-name>)`

$V^{c}_{constraints}$ `::= [set | list | bag | union] {<derived-type-name>}`

Example 5.24: In the CPS, each `Paper` has `Keywords` that may contain 1 to 5 entries, with no duplicates.

$V^{c}_{constraints}$ `= Paper ::= set (Keywords) , MinLen = 1, MaxLen = 5`

Example 5.25: In the CPS, the conceptual view `Author-Index` is of list type as it is an ordered collection of author names and allows duplicates as two or more authors may have similar names.

$V^{c}_{constraints}$ = `Author-Index ::= list (lastName, firstName)`

Example 5.26: In the CPS, the `paperSubmission` date may be specified using typical date format or using week numbers.

$V^{c}_{constraints}$ = `paperSubmissionDate ::= union (date, number (99))`

5.6 Conclusion

In this chapter, we present the core theory and concepts that are relevant to the views in the LVM and their properties. Firstly, we present a formal semantics and definition of views at different levels of abstraction, namely conceptual, logical and document, including formal description of the transformation between these layers of abstraction. Secondly, we provided a definition for specifying views in the LVM that combines different abstraction layers. Thirdly, we provided formal semantics and a detailed description of the conceptual view operators that enable one to construct conceptual views, at the conceptual level. Finally, we provided semantics for modelling and specifying constraints associated with the views in the LVM.

In the following chapters, namely Chapters 6 and 7, we will provide detailed discussions of modelling and transforming views in the LVM, using two OO modelling languages with the help of an illustrative industrial case study example. The formal semantics described in this chapter is utilized in illustrating the LVM properties and its applications. And later in Chapters 8 to 10, the concepts presented in this chapter is utilized (or extended) to develop architectural framework for three real-world data-intensive application scenarios.

References

ANSI & ISO 2003, *ANSI - SQL 2003*, ANSI / ISO.

Chen, YB, Ling, TW & Lee, ML 2002, 'Designing Valid XML Views', *Proceedings of the 21st International Conference on Conceptual Modeling (ER '02)*, vol. 2503, eds S

Spaccapietra, ST March & Y Kambayashi, Springer-Verlag London, UK, Tampere, Finland, pp. 463 - 78.

Dillon, TS & Tan, PL 1993, *Object-Oriented Conceptual Modeling*, Prentice Hall, Australia.

Doorn, JH, Rivero, LC & (eds) 2002, *Database Integrity: Challenges & Solutions*, eds Jorge H. Doorn & LCR (eds), Idea Group Publishing, Hershey, PA.

Elmasri, R & Navathe, S 2004, *Fundamentals of database systems*, 4th edn, Pearson/Addison Wesley, New York.

Feng, L, Chang, E & Dillon, TS 2002, 'A Semantic Network-based Design Methodology for XML Documents', *ACM Transactions on Information Systems (TOIS)*, vol. 20, No 4, no. 4, pp. 390 - 421

Feng, L, Chang, E & Dillon, TS 2003, 'Schemata Transformation of Object-Oriented Conceptual Models to XML', *International Journal of Computer Systems Science & Engineering*, vol. 18, No. 1, no. 1, pp. 45-60

Golfarelli, M, Maio, D & Rizzi, S 1998, 'The Dimensional Fact Model: A Conceptual Model for Data Warehouses', *International Journal of Cooperative Information Systems*, vol. 7, no. 2-3, pp. 215-47

OMG-UML™ 2003, *UML 2.0 Final Adopted Specification (http://www.uml.org/#UML2.0)*, OMG, 2005.

Trujillo, J, Palomar, M, Gomez, J & Song, I-Y 2001, 'Designing Data Warehouses with OO Conceptual Models', *IEEE Computer Society, "Computer"*, December, 2001, pp. 66-75.

Ullman, JD 1989, *Principles of database and knowledge-base systems*, vol. 1, Principles of computer science series., Computer Science Press, Rockville, Md.

W3C-XPath 1999, *XML Path Language (XPath) Version 1.0 (http://www.w3.org/TR/xpath/)*, The World Wide Web Consortium (W3C), November 1999.

W3C-XqFM 2005, *XQuery 1.0 and XPath 2.0 Formal Semantics (http://www.w3.org/TR/xquery-semantics/)*, The World Wide Web Consortium (W3C), viewed November 2005.

W3C-XQuery 2004, *XQuery 1.0: An XML Query Language (http://www.w3c.org/)*, The World Wide Web Consortium (W3C), viewed November 2003 http://www.w3.org/TR/xquery.

W3C-XSD 2001, *XML Schema (http://www.w3.org/XML/Schema)*, W3C, 2004.

Wouters, C 2006, 'A Formalization and Application of Ontology Extraction', Doctor of Philosophy (Ph.D) thesis, La Trobe University, Melbourne, Australia, Melbourne.

Wouters, C, Dillon, TS, Rahayu, JW, Chang, E & Meersman, R 2004, 'A Practical Approach to the Derivation of a Materialized Ontology View', in D Taniar & W Rahayu (eds), *Web Information Systems*, Idea Group Publishing, USA.

Xiaou, R, Dillon, TS, Chang, E & Feng, L 2001a, 'Mapping Object Relationships into XML Schema', *Proceedings of OOPSLA Workshop on Objects, XML and Databases*.

Xiaou, R, Dillon, TS, Chang, E & Feng, L 2001b, 'Modeling and Transformation of Object-Oriented Conceptual Models into XML Schema', *12th International Conference on Database and Expert Systems Applications (DEXA '01) 2001*, vol. 2113, ed. JL Heinrich C. Mayr, Gerald Quirchmayr, Pavel Vogel, Springer.

Chapter 6

Modelling and Transformation I

6.1 Introduction

In this chapter, we continue our discussion from Chapters 4 and 5 on the layered view model. Here, we focus on modelling and transformation of conceptual views to logical and document level views. We present modelling and transformation conceptual views that are specified in XSemantic nets. XSemantic net based design transformation methodology was developed originally by Feng et al. in (Feng, Chang & Dillon 2002) to enforce semantically rich conceptual modelling semantics for XML documents. Thus, it was intended for conceptual modelling of XML domains using OO concepts. It is based on the graph theory and cablesets. Here, we extend the design methodology to model conceptual views (including both view data and behavior) using OO concepts.

To illustrate our concepts in this chapter, we gradually model the example e-Sol case study described in Chapter 4.

The rest of the chapter is organized as follows. A brief overview of modelling issues related to the conceptual views in the LVM is given in section 6.2. Section 6.3 presents the XSemantic net notations and model semantics in detail, including some of theoretical concepts and the extensions provided from the original modelling notation presented in (Feng, Chang & Dillon 2002). In section 6.4, a detailed discussion on modelling conceptual views using XSemantic net notation and semantics is given, followed by section 6.5, which provides a detailed discussion on schemata transformation conceptual views specified in XSemantic nets to logical views is provided. Section 6.6 provides discussion of modelling and representing conceptual dynamic properties using XSemantic nets. This is followed in section 6.7 by a discussion of the transformation of such dynamic view properties to document view expressions. Section 6.8 concludes this chapter.

6.2 Modelling Conceptual Views

Conceptual views are views that are captured at the conceptual level with conceptual semantics and constraints. To the best of our knowledge, there exists no notation to specify or define such high level views or view semantics using existing OO modelling languages. Even in the ORA-SS view model (Chen, Ling & Lee 2002), the views are represented (in contrast to defining) using only a diagrammatic notation.

Also, it should be noted that, due to its abstract nature, conceptual views may be modelled using any high level modelling languages that support OO concepts, such as:

(i) Dillon & Tan notation (Dillon & Tan 1993),

(ii) Object Role Modeling (ORM)[1]

(iii) OMG's UML (OMG-UML™ 2003b)

(iv) XSemantic Nets

(v) Enhanced-ER (Enhanced or Extended Entity-Relationship Model (EER)) (Elmasri & Navathe 2000).

Here in our research, we demonstrate the use of language notations (iii) and (iv) above, as they are well understood and there exists a basic set of well-defined and easily extensible schemata transformation rules to transform conceptual models to schema models, namely XML Schema.

(i) Thus, in the layered view model, to define and specify views at the conceptual level (*conceptual views*), a new set of notations needs to be developed. These include:

[1] http://www.orm.net/index.html

(ii) a notation to represent conceptual views, visually, at the conceptual level;

(iii) a set of notations to represent conceptual view properties;

(iv) a notation to represent view constructs;

(v) a notation to represent conceptual view constraints; and

(vi) a notation to represent conceptual view dynamic properties.

As stated earlier in Chapter 3, the two OO modelling languages considered in this research to model conceptual views are: (a) XSemantic net (Feng, Chang & Dillon 2002) and (b) OMG UML (OMG-UML™ 2003a) (with OCL (OMG-OCL 2003)). We first look at modelling issues with XSemantic nets in detail here, and in the following chapter we consider UML in detail.

6.3 XSemantic Nets

As stated in Chapter 4, XSemantic nets provide three design levels to model a given XML domain in the required level of detail. Here, we discuss modelling conceptual views using XSemantic nets.

The advantages of using XSemantic net for conceptual views include:

(i) XSemantic net modelling notations are structurally similar to XML documents, thus they are easier to understand and model complex document structures;

(ii) based on OO concepts thus provide semantically rich modelling constructs;

(iii) unlike other graph based modelling notations (e.g. such as ORM) XSemantic net utilizes *cableset* theory to preserve OO classes, attributes and class hierarchies (Feng, Chang & Dillon 2002);

(iv) support an extensive set of semi-structured data specific, visual constraint specifications;

(v) due to its structural similarity to XML, transformation of XSemantic net (conceptual) models to XML schema and/or other specifications (e.g. XMI) are easier and straightforward (two-step transformation);

(vi) though it is based on acyclic graphs, the modelling notation is easily extended to add new concepts and notations such as conceptual views (discussed here), ontology (discussed in (Wouters 2006) and briefly in Chapter 9 etc.).

XSemantic nets provide semantic modelling of XML domains through five major components which were textually presented in Chapter 4 and here are elaborated with mathematical concepts and symbols. They are: (Feng, Chang & Dillon 2002);

(1) a set of *nodes* N_{ode}, representing real-world *objects* and *view* objects,

   a. basic nodes representing simple content, $s_{content}$

   b. complex nodes representing complex content, $c_{content}$

(2) a set of *event nodes* representing *generic* and *user defined* methods (or triggers) in the *view* objects;

(3) a set of *directed labelled edges* E_{dge}, representing *semantic relationships* between these objects;

(4) a set of *labels* L_{able}, denoting different types of semantic relationships, including:

   a. aggregation,

   b. generalization,

   c. association,

   d. of-property

   e. view

(5) a set of *constraints*,

a. defined over a node (e.g. uniqueness, referential integrity, etc.)

b. defined over an edge (e.g. cardinality, adhesion, etc.)

c. defined over a set of edges (e.g. exclusive disjunction, order, etc.)

(6) a set of *conceptual operators* to systemically construct conceptual views

a. unary operators

b. binary operators

Here, it should be noted that in the original work by authors in (Feng, Chang & Dillon 2002), the abovementioned concepts in points (1), (3) (4) and (5) were presented in detail. In this research, we have extended these concepts to support the notation for conceptual views. The concepts in points (2) and (6) are extensions proposed as part of this research to the original XSemantic net work in (Feng, Chang & Dillon 2002).

XSemantic net nodes correspond to two types of content namely: (a) *simple content* $s_{content}$ and (b) *complex content* $c_{content}$. Simple contents correspond to basic nodes and complex contents correspond to complex nodes. However, as stated earlier, an *event node* is not part of the original XSemantic net concepts and is classified here as a special type of complex content and will be described later in this section.

The simple content of a basic node may be:

(i) an atomic value: this corresponds values from the domains of basic data types such as string, integer, character etc.

(ii) a constructional value: this corresponds to values such as set, list, bag or union values, as described in Chapter 5. Here, except for the union type, all members in the constructional value must be of same type.

a. A *set* value is an unordered collection type without duplicates. Here, all members are unique and of the same type.

b. A *list* value is an ordered collection type with duplicates.

c. A *bag* value is an unordered collection type with duplicates (i.e. bag a set value with duplicates).

d. A *union* value is a combination of one or more types combined to provide more than one value for a given type. One as one type.

Let $dt_1, dt_2, ..., dt_n$ be a series of basic data types with the corresponding domains denoted as $dom(dt_1), ..., dom(dt_n)$. Let content of a basic node be denoted as c_{ontent}. It should be noted that the definitions below are similar or analogous or extensions to the ones presented in the original XSemantic net work in (Feng, Chang & Dillon 2002).

Definition 6.1. A basic node is said to be having an *atomic value* if and only if $c_{ontent} \in \bigcup_{i=1}^{n} dom(dt_i)$.

Definition 6.2. A basic node is said to be have a *set value* if and only if $c_{ontent} = \{c_1, c_2, ...c_k\}$, where $\exists i (1 \leq i \leq n) c_i \in dom(dt_i)_j)$ $\wedge$ $\forall i \forall j (1 \leq i, j \leq k)(i \neq j \Leftrightarrow c_i \neq c_j)$.

Definition 6.3. A basic node is said to be having a *bag value* if and only if $c_{ontent} = \{c_1, c_2, ...c_k\}$, where $\exists i (1 \leq i \leq n) c_i \in dom(dt_i)$.

Definition 6.4. A basic node is said to be having a *list value* if and only if $c_{ontent} = \{c_1, c_2, ...c_k\}$, where $\exists i(1 \le i \le n)c_i \in dom(dt_i)$ and all the values in c_{ontent} are in a prescribed order.

Complex content $c_{content}$ of a complex node refers to some other nodes through directed labelled edges, where each edge connects a pair of nodes with the relationship showed by the label of the edge. One of the unique features of XSemantic net is that, unlike similar notations such as Object Role Modeling (ORM), the definition of the complex content is defined using *connection cluster set*, a notion defined using the ***cableset*** approach (Feng, Chang & Dillon 2002).

Definition 6.5. A *connection* of a node $n \in N_{ode}$ is an ordered pair (r, n_1) such that, r is a label in $r \in L_{able}$ and n_1 is a node in $n_1 \in N_{ode}$, representing that node n is connected to n_1 via relation r. (Feng, Chang & Dillon 2002)

Definition 6.6. A *connection cluster* of a node $n \in N_{ode}$ is an ordered pair (r, ns) such that, r is a label in $r \in L_{able}$ and ns is a set of node in $ns = \{n_1, ...n_j...\}, n_j \in N_{ode}$, representing that node n is connected to each node ns via relation r. (Feng, Chang & Dillon 2002)

Definition 6.7. A *connection cluster set* of a node $n \in N_{ode}$ is a set of connection clusters such that, $\{(r_1, ns_1), ...(r_k, ns_k)\}$ where $\forall i \forall j\,(1 \le i, j \le k)$ $(i \ne j \Leftrightarrow r_i \ne r_j)$ (Feng, Chang & Dillon 2002).

Definition 6.8. A *complex content* of a complex node is a connection cluster set. (Feng, Chang & Dillon 2002)

Now we present the notional definitions of XSemantic net nodes, labels and edges.

Definition 6.9. A node $n \in \mathrm{N}_{ode}$ is a 4-tuple consisting of a unique node identifier n_{id}, a node name n_{name}, a node category $n_{category}$ indicating whether the node is a basic, complex, or event and node content $n_{content}$.

$$n = (n_{id} , n_{name} , n_{category} , n_{content}) \qquad \textbf{(6.1)}$$

An n-category specification of the node can have a simple or complex content. If $n_{category}$ is "*basic*" it is said to have simple content, and it "*complex*" for complex content and "*event*" for conceptual constructs or events.

Note: In order to make it easier to follow, from here on we refer to the nodes using their node name (label).

Given a simple node n_s and complex node n_c $(n_s, n_c \in \mathrm{N}_{ode})$, the following holds true:

$$n_s \cap n_c = \phi \text{ and } n_s \cup n_c = \mathrm{N}_{ode} \qquad \textbf{(6.2)}$$

In the XSemantic net, labels are a set of enumerated string types indicating semantic relationships, dependency and views. Thus, a label L_{able} is a set such that:

$$L_{able} = \{g, s, a, p, v\} \qquad (6.3)$$

where, the symbols g, s, a, p and v to denote:

(i) g = **g**eneralization / specialization relationship

(ii) s = a**s**sociation relationship

(iii) a = **a**ggregation or part-of relationship

(iv) p = of-**p**roperty (or dependency) relationship

(v) v = **v**iew dependency

Definition 6.10. An edge $e \in E_{dge}$ is a 3-tuple consisting of a label $e_{label} \in L_{able}$ stating the edge type, the source node of the edge $e_{source-node} \in N_{ode}$, and the target node of the edge $e_{target-node} \in N_{ode}$.

$$e = (e_{label}, e_{source-node}, e_{target-node}) \qquad (6.4)$$

where,

$$e_{label} \in L_{able} \text{ and } e_{source-node}, e_{target-node} \in N_{ode} \qquad (6.5)$$

Alternatively, an edge may be shown as:

$$e_{source-node} \xrightarrow{e_{label}} e_{target-node} \qquad (6.6)$$

Definition 6.11. An edge $e_{source-node} \xrightarrow{e_{label}} e_{target-node}$ is said to be well-formed, if and only if the following statements hold true:

(i) if e_{label} = "**p**", the source $e_{source-node}$ is always a complex node and the target $e_{target-node}$ is always a simple node. That is:

$$(e_{label} = p) \Rightarrow (e_{source-node} \in n_c \wedge e_{target-node} \in n_s) \qquad \textbf{(6.7)}$$

(ii) if e_{label} = "**g**", "**s**" or "**a**" the source $e_{source-node}$ is always a complex node.

$$(e_{label} \in \{g, s, a\}) \Rightarrow (e_{source-node} \in n_c) \wedge (e_{target-node} \in \mathrm{N}_{ode}) \qquad \textbf{(6.8)}$$

(iii) if e_{label} = "**v**", the source $e_{source-node}$ is an event node. Let n_e be an event node $(n_e \in \mathrm{N}_{ode})$. Thus, $n_e \cup n_s = \mathrm{N}_{ode}$ and also $n_e \cup n_c = \mathrm{N}_{ode}$. Therefore,

$$(e_{label} \in v) \Rightarrow (e_{source-node} \in n_e) \wedge (e_{target-node} \in \mathrm{N}_{ode}) \qquad \textbf{(6.9)}$$

(iv) if e_{label} = "**op**", the source $e_{source-node}$ is a set of complex nodes and the target node is an event node n_e $(n_e \in \mathrm{N}_{ode})$. Let $N_c = \{n_c^1, n_c^2, .. n_c^k, ..\}$ be a set of complex nodes, where $n_c^k \in n_c$. Therefore:

$$(e_{label} \in op) \Rightarrow (e_{source-node} \in N_c) \wedge (e_{target-node} \in n_e) \qquad \textbf{(6.10)}$$

In this research, all edges under discussion are assumed to be well-formed. Furthermore, as defined in Chapter 3, concepts such as *path*, *length* and *depth* can be defined the XSemantic net.

Let γ_{path} denote a path with starting node n_1 vertex and ending node n_k of a sequence of connected nodes $n_1, n_2, ...n_i, ...n_k$.

Definition 6.12. The path γ_{path} is a *sequence* of nodes with a *start* node, an *end* node, and for any adjacent vertices, there exists a well-formed edge.

$$\gamma_{path} \mapsto (n_1, n_2, ...n_i, ...n_k) \qquad \textbf{(6.11)}$$

where

$$n_i \in \mathrm{N}_{ode} \text{ and } 1 \le i \le k \qquad \textbf{(6.12)}$$

For any two *consecutive* vertices n_i and n_{i+1} in the path γ_{path} for $1 \le i \le k-1$:

$$(l, n_i, n_{i+1}) = e_i \qquad \textbf{(6.13)}$$

where $e_i \in \mathrm{E}_{dge}$.

Also, from Chapter 3, using equation 3.45, it can be shown that, given a sequence of edges $(e_1, e_2, ...e_j, ...e_n)$, $(e_i \in \mathrm{E}_{dge})$ a path γ_{path} may be written as:

$$n_t(e_j) = n_s(e_{j+1}), \text{ where } 1 \le j \le n \qquad (\mathbf{6.14})$$

where $n_s(e_i), n_t(e_i) \in \mathrm{N}_{ode}$.

Conversely, a path γ_{path} can be also stated as, the path γ_{path} that goes through node n_i $(\in \mathrm{N}_{ode})$, where $1 \le i \le n$. Let us denote the length of a path as $length(\gamma_{path})$.

Definition 6.13. The length of a path $length(\gamma_{path})$ is defined as the number of nodes that a path goes through, which is equal to the total number of nodes between the start and end nodes, inclusive.

$$length(\gamma_{ath}) = \sum_{i=1}^{n} n_i \qquad (\mathbf{6.15})$$

Definition 6.14. A path γ^{o}_{ath} is said to be an *original* path, if and only if, the start node n_1 has no incoming edges. That is, the root node of the XSemantic net.

In simple terms, an original path usually has the root node as its starting node. Thus, we can show a root node in XSemantic net as:

Definition 6.15. The root is the *start* node of an original path γ^{o}_{ath} such that the path length is 1 (i.e. $length(p^{o}_{ath}) = 1$).

Let node $n \in N_{ode}$ be the *end* node to a collection of original paths $\gamma^{o}_{1}, \gamma^{o}_{2}, \ldots\ldots, \gamma^{o}_{i}, \ldots \gamma^{o}_{s}$.

Definition 6.16. The depth of a node n is the maximal length of the original path in $\gamma^{o}_{1}, \gamma^{o}_{2}, \ldots\ldots, \gamma^{o}_{i}, \ldots \gamma^{o}_{s}$.

$$depth(n) = Max^{s}_{i=1}(\gamma^{o}_{i}) \qquad \textbf{(6.16)}$$

Also, from equation 3.10, a root node $n_{root} \in N_{ode}$ in XSemantic net may be stated as:

Definition 6.17. The root node is a node with the node depth of 1 (i.e. $depth(n_{root}) = 1$).

6.4 Modelling Conceptual Views

In this section we present modelling and specifying conceptual views using XSemantic nets.

6.4.1 XSemantic Net Notation

As described in Chapters 3 and 4, the *extended* XSemantic net notation used in this research is given below in Fig.6.1. The rest of this section describes how conceptual view properties are represented in the extended XSemantic nets.

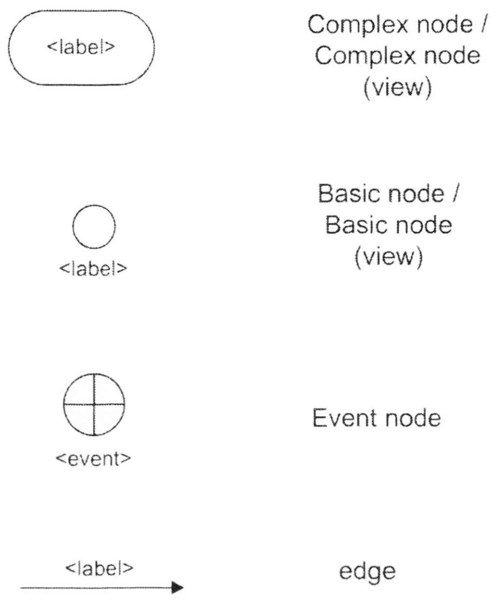


Figure 6.1: XSemantic net Notation

6.4.1.1 Conceptual Constructs

To show the relationship between a conceptual view and the stored objects (context) from which it is constructed, we use directed edges with the label "**op**". This is shown in Fig. 6.2. Here, conditions shown in equations 6.9 and 6.10 hold true.

6.4.1.2 Conceptual Views

Representing conceptual view in XSemantic nets is similar to representing real-world objects as described in (Feng, Chang & Dillon 2002) except conditions given in equations 6.9 and 6.10 above apply. That is, the root node of a conceptual view will originate from an event node, which in turn represents the conceptual operator (or conceptual construct). This is shown in Fig. 6.2.

Example 6.1: In the case of the conceptual view `Income` (shown in Fig. 6.2), the conceptual construct is a conceptual JOIN operator with join conditions, where x = `Staff`, y = `Salary-Pkg` and z = `Benefit-Pkg`:

$$\triangleright\triangleleft_{(x,y,z)} = (x \rightarrow_{(x.staffID=y.staffID)} y) \ and \quad (x \rightarrow_{(x.staffID=z.staffID)} z)$$

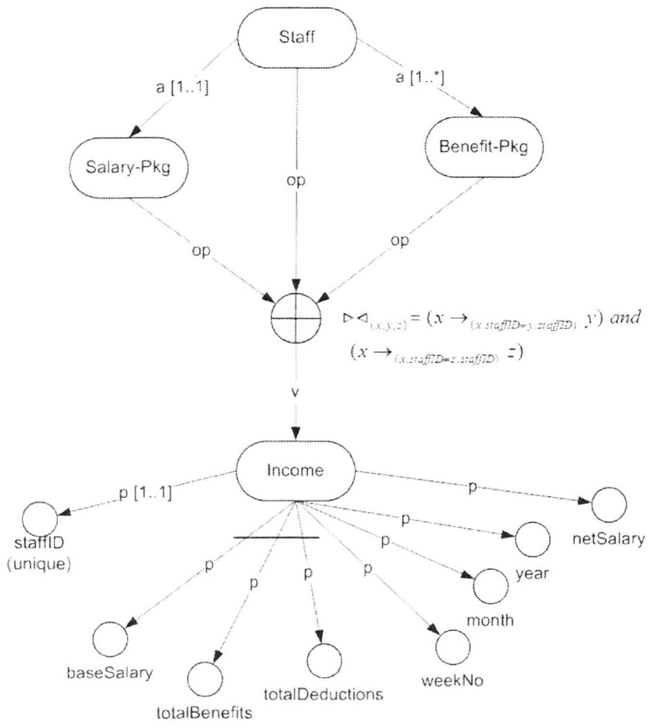


Figure 6.2: A conceptual view example in the e-Sol ("Income")

6.4.1.3 OO Semantic Relationships

As described in (Feng, Chang & Dillon 2002) and Chapter 3, in XSemantic nets, semantic relationships among real-world objets are shown using directed edges with labels and optional constraints. Similarly, semantic relationships between one or more conceptual views are also shown using the same notations. Some examples of OO relationships in the e-Sol case study are shown in Figs. 6.3 and 6.4. There are four types of relationships in XSemantic nets.

Of-property relationship in XSemantic net links a property (i.e. an attribute) with its object. As described in Chapter 3, an aggregation relationship is a hierarchical relationship that organizes objects in an object hierarchy, the "whole" object consisting of "part" objects. By its very nature, XML (and XSemantic net) is hierarchical and represents a logical hierarchy between its node elements and sub-elements. An association relationship specifies that an object of one kind is associated with the objects of another, in a non-hierarchical manner (Dillon & Tan 1993). That is, both participating objects in the association relationships are at the same level. The generalization/specialization relationship is an important concept in OO, as well as for conceptual views. One or more specialized views (descendant view) can be derived (by extension or by restriction) from an existing conceptual view (ancestor view(s)).

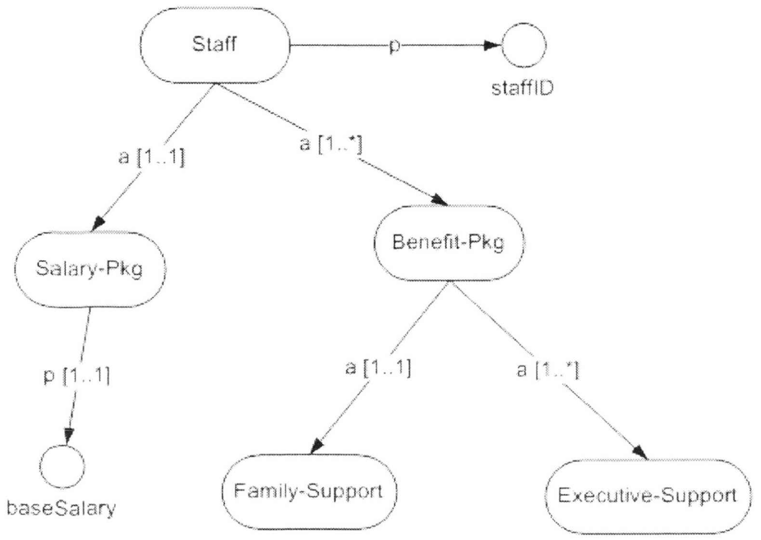


Figure 6.3: Conceptual view semantic relationship examples (aggregation, of-property)

Example 6.2: There exists an aggregation relationship (Fig. 6.3) between the whole-object `Staff` and part-objects `Salary-Pkg` and `Benefit-Pkg`. Also, `Family-Support` and `Executive-Support` are part-objects of the whole-object `Benefit-Pkg` in the aggregation hierarchy.

Example 6.3: There exists an object-attribute relationship (i.e. of-property) between `Staff` and `staffID`. As shown in Fig. 6.3. A similar relationship exists between `staffID`, `baseSalary`, `totalBenefits` etc. of the conceptual view `Income` as shown in Fig. 6.2.

Example 6.4: The relationship between conceptual view `Logistics-Staff`, `Admin` and `Site-Manager` is generalization/specialization, as shown in Fig. 6.4.

Example 6.5: The relationship between conceptual views `Site-Manager` and `Warehouse-Manager` is association, as shown in Fig. 6.4. `Site-Manager` *manage* `Warehouse-Manager` managers.

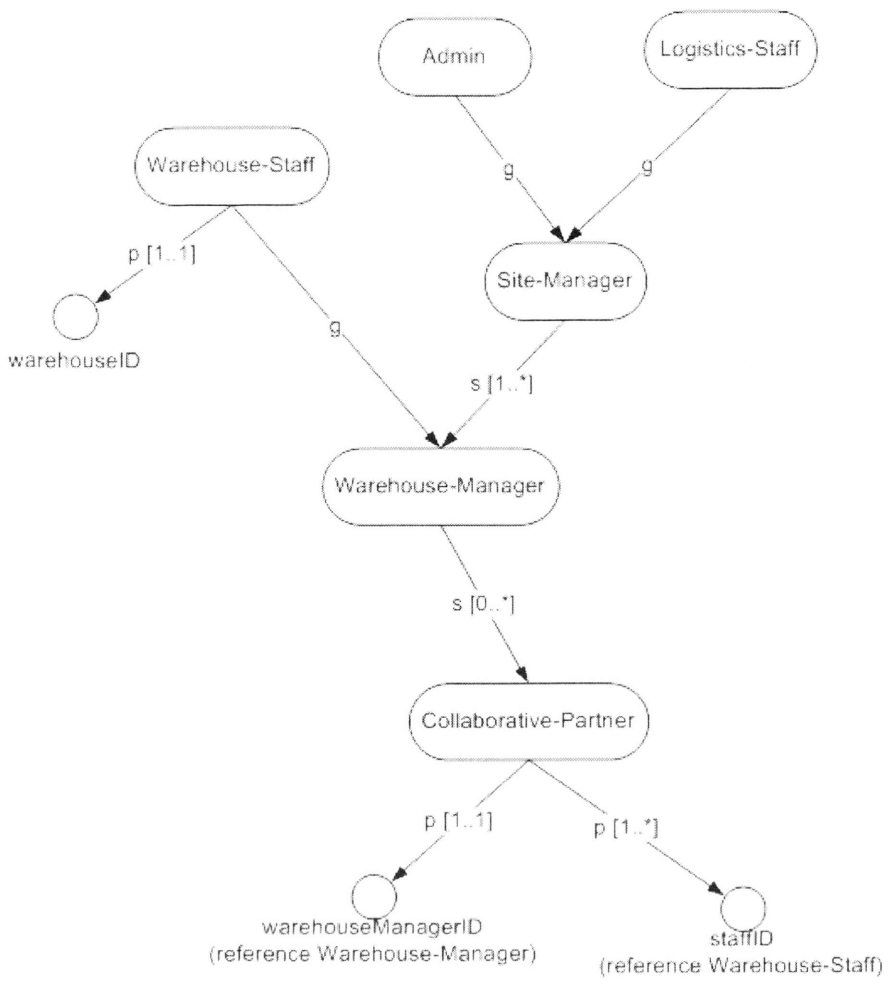


Figure 6.4: OO conceptual view semantic relationship examples (generalization, association)

Example 6.6: The relationship between conceptual views `Warehouse-Manager` and `Collaborative-Partner` is association, as shown in Fig. 6.4. `Warehouse-Manager` ***deals-with*** `Collaborative-Partner`.

Example 6.7: The relationship between conceptual view `Warehouse-Staff` and `Warehouse-Manager` is generalization/specialization, as shown in Fig. 6.4.

6.4.2 Conceptual View Constraints

This section presents how conceptual view constraints are visually specified in XSemantic nets. In XSemantic nets, constraints may be grouped into three broad categories, namely:

(i) constraints defined over an edge

(ii) constraints defined over a set of edges

(iii) constraints defined over a node

6.4.2.1 Constraints Defined over an Edge

Constraints defined over an edge usually take the form of textual description as part of the label of the edges. There are two types of such constraints namely, cardinality constraints and homogenous/heterogenous constraints.

(i) Cardinality Constraint

The cardinality constraint shows the number of instances of one conceptual view that may relate to a single instance of another. This is similar to that of directional OO (binary) relationships and is shown as part of the labelled edges. Table 6.1 summarises possible cardinalities for various relationships, and in XSemantic net, it is in the form of: `<relationship-name> :: [n..m],` where the possible values of `n` & `m`, and their meaning is given in Table 6.1 below.

Table 6.1: Relationships and cardinality constraints

Relationship name	n	m	Cardinality	Adhesion
association	0	1	Zero-or-one	weak
association	0	*	Zero-or-more	weak
association, aggregation, of-property	1	1	One-to-one	strong
association, aggregation, of-property	1	*	One-to-many	strong or weak
association	*	1	Many-to-one	strong or weak
association	*	*	Many-to-many	strong or weak

(ii) Homogenous Composition

In an aggregation relationship, if part-object/(s) is of the same type as the whole-object, it is described as a homogenous composition. This can be shown as a typical aggregation relationship, with keyword "`homogenous`" stated in the labelled edge.

Example 6.8: In the e-Sol case study, `Family-Support`, `Executive-Support` are of type `Benefit-Pkg`. This is shown in Fig.6.5.

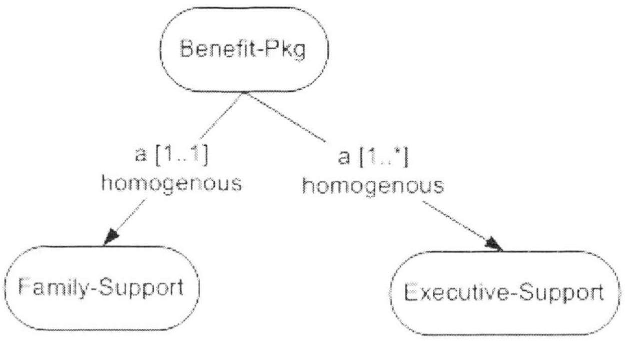


Figure 6.5: Homogenous composition example

Example 6.9: Also, as shown in Fig.6.18, `Goods-Sub-Type` is of type `Goods-Type`.

(iii) Dependency / Adhesion Constraint

To represent adhesion (or dependency) constraints, a labelled edge is used with the keyword "`strong adhesion`" or "`weak adhesion`" as shown in Fig. 6.6. In this research, unless explicitly stated, all adhesive relationships in the XSemantic nets (i.e. association, aggregation, of-property) are considered as *strong* adhesion and the view relationship is always considered as *weak* adhesion.

Table 6.1 shows various adhesion constraints based on cardinality constraints for various relationships.

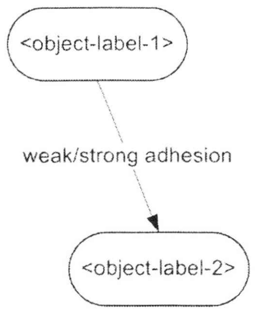


Figure 6.6: Adhesion constraint

Example 6.10: In the case of conceptual views `Lot-Movement-History`, `Internal-Lot-Movement-History` and `External-Lot-Movement-History`, the relationship has weak adhesion as shown in Fig. 6.7, as `Lot-Movement-History` may have either internal or external lot movement histories.

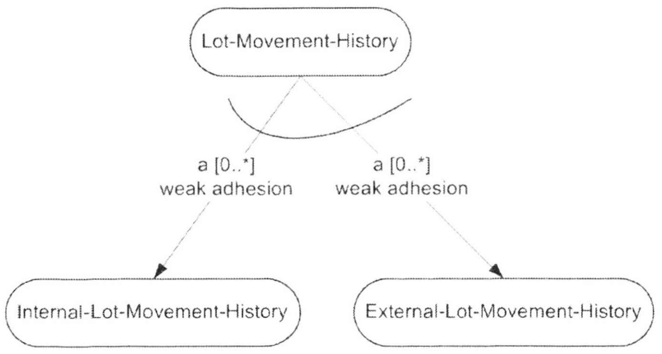


Figure 6.7: Adhesion constraint example

Example 6.11: In the case of conceptual view `Warehouse-Manager` the attribute (i.e. of-property relationship) `staffID` is compulsory and shown as strong adhesion in Fig. 6.8(a).

Example 6.12: Conversely, in the case of conceptual view `Goods-Sub-Type` the attribute `goodsDescription` is an optional *set* value with zero or more entries and shown as weak adhesion in Fig. 6.8(b).

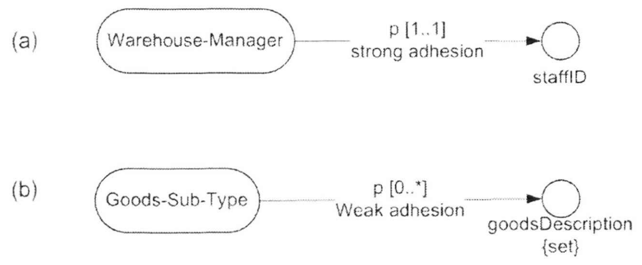


Figure 6.8: Adhesion constraint examples

6.4.2.2 Constraints Defined over a Set of Edges

(i) Ordered Composition

To visually and explicitly state ordered composition for conceptual views using XSemantic nets, one can use a line across the outgoing edges of the whole object as shown in Fig. 6.9. The order is designated as being left to the line or along the line, i.e. the left most object or node on the line is the first and the right most object is the last in the ordered composition.

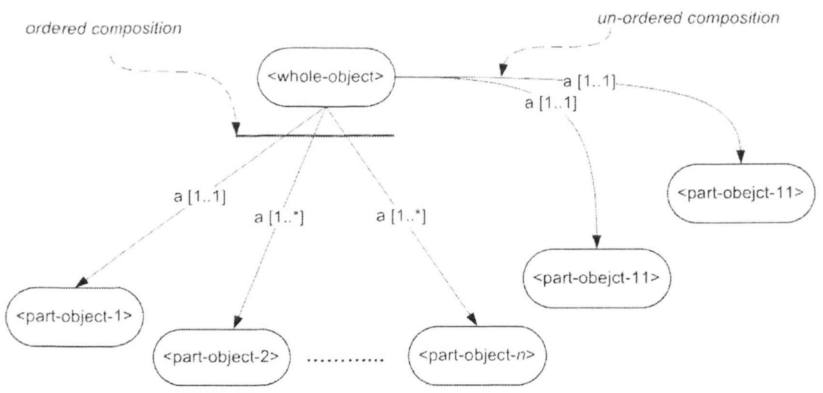


Figure 6.9: Ordered-Composition

In addition, for an ordered composition to be valid for conceptual views, the condition stated in equation 6.8 above should hold true (i.e. $e_{label} = a$).

Example 6.13: In the e-Sol. example, in the composition relationship between whole-object `Lot-Movement-History` and the part-objects (`Internal-Lot-Movement-History`, `External-Lot-Movement-History`) is *ordered*. This is shown in Fig. 6.10.

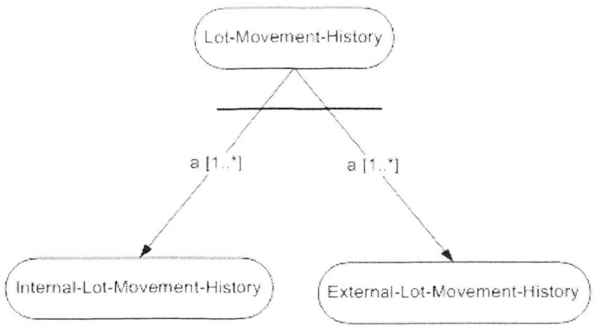


Figure 6.10: Ordered-Composition example

(ii) Exclusive Disjunction

Similar ordered-composition, in XSemantic nets, an arc is used to visually show exclusive disjunction between two objects. Exclusive disjunction is allowed only between association and aggregation relationships. That is, in equation 6.8, either $e_{label} = s$ or $e_{label} = a$. This is shown in Fig 6.11.

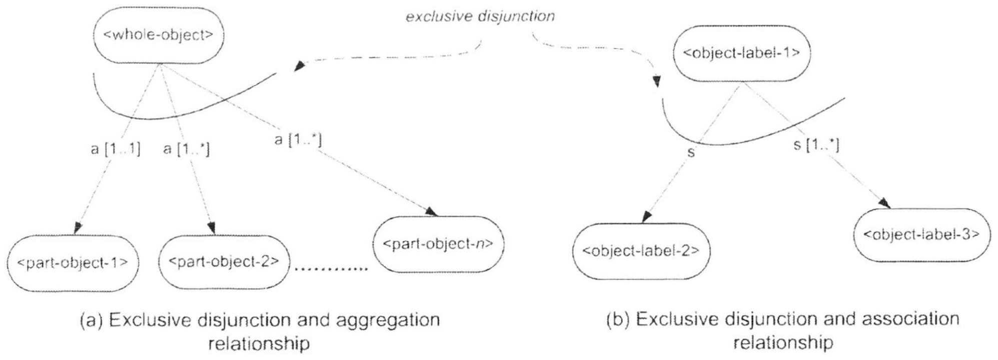


Figure 6.11: Exclusive disjunction

Example 6.14: In the case of conceptual views `Lot-Movement-History` (e-Sol), the exclusive disjunction between `Internal-Lot-Movement` (stored goods change owners) and `External-Lot-Movement` (goods shipped outside the warehouse) can be shown via an arc in XSemantic net (as shown in Fig. 6.12).

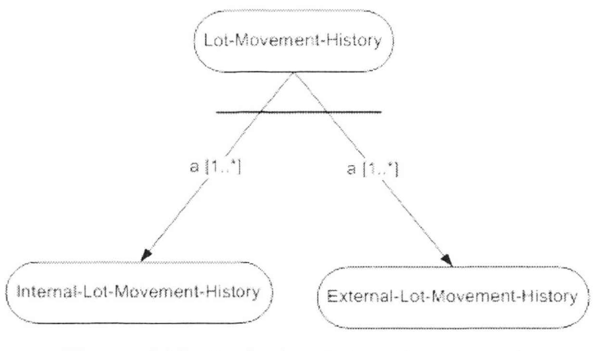


Figure 6.12: Exclusive disjunction example

6.4.2.3 Constraints Defined over a Node

(i) Unique Constraint

Unique constraints (e.g. OID) in XSemantic nets are visually (and explicitly) shown by a keyword "`unique`" as part of the node label. This is shown in Fig. 6.13. An example is given in Fig. 6.15.

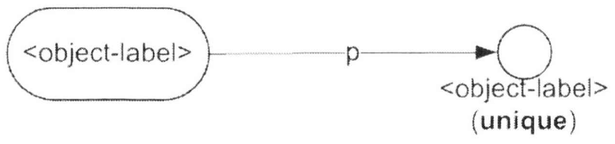


Figure 6.13: Unique constraint specification

(ii) OID Constraint

In many OO conceptual models and diagrams (including earlier versions of XSemantic and the present UML 2.0), though the concept of Object Identifier (OID, a unique concept to OO systems) is assumed and never visually represented as part of the conceptual model (unlike primary keys in E/ER), here, in our work with *conceptual views*, we need to explicitly state the OIDs and these should be available to visualize at the highest level of abstraction. Therefore, here, we provide a means of using OIDs for the purpose of IDs, similar to that of primary/foreign key constraints available in E/ER models. We argue that just utilizing OID in our conceptual model provides additional semantics, such as providing Id/keys, referential and integrity constraints that are visually lacking in many OO conceptual modelling technique.

To model OID is XSemantic nets, since it is a *unique* constraint by nature, we use the keywords "OID" as part of the node label, as shown in Fig. 6.14. It should be noted that:

(i) for a given (complex) node only *ONE* OID constraint is possible

(ii) it may have one or more unique constraints associated with its (basic) node

(iii) by specifying OID constraints, it also implies unique constraint.

(iv) the adhesive relationship (described in section 6.4.2.1) is always *strong* between an object and its OID attribute.

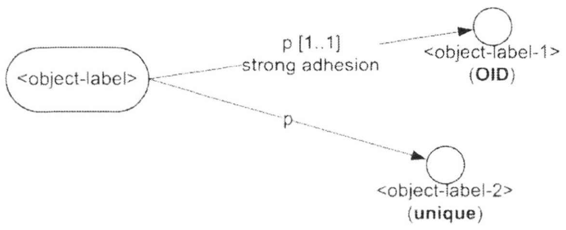


Figure 6.14: OID, unique constraint specification

Example 6.15: In the case of conceptual view Logistics-Staff (Fig. 6.15), managedBy attribute is unique, as each logistics staff reports to only one site manager. But Logistics-Staff has a unique OID, staffID.

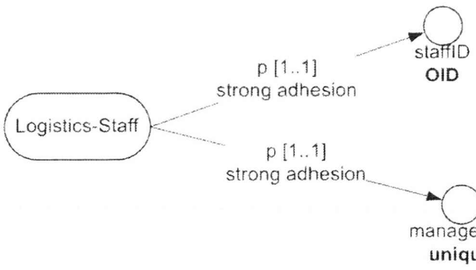


Figure 6.15: Unique constraint example

Example 6.16: Each warehouse (including Warehouse-Manager) has a unique OID, staffID. This is shown in n Fig. 6.16.

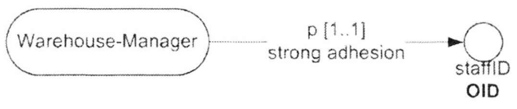


Figure 6.16: OID constraint example

(v) Referential Constraint

Referential constraints in XSemantic nets are visually (and explicitly) shown by the keyword "`reference`" together with the referenced object name as shown in Fig. 6.17.

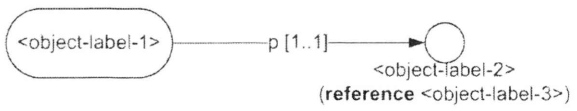


Figure 6.17: Referential constraint

Example 6.17: The reference between `Goods-Items` and `Goods-Sub-Type`, that is, in the case of `Goods-Items` that are of certain type as defined `Goods-Sub-Type` is visually shown in XSemantic as shown in Fig.6.18. Another example is given in Fig. 6.4 above.

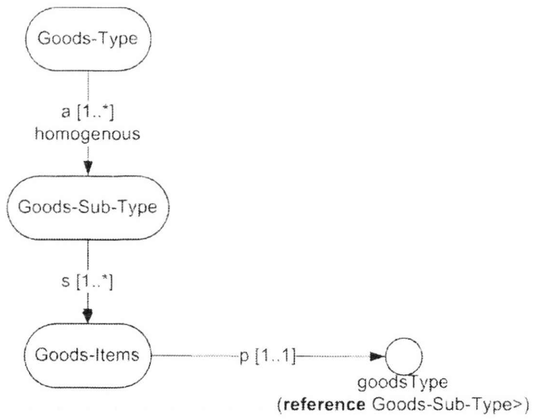


Figure 6.18: Example of Composition and association relationships (with referential constraint)

(vi) Constructional Constraints

As described earlier, simple content shown as basic nodes may have constructional values such as set, list, bag or union. In XSemantic nets, constructional constraints refer to additional descriptions defined for basic node types to indicate such non-atomic values that are allowed (e.g. set, bag, or list). In order to show these constructional constraints, textual labels are placed in addition to the basic node labels, as shown in Fig. 6.19.

It should be noted that, as described in Chapter 5, the *union constraint* is different from the *union conceptual operator*.

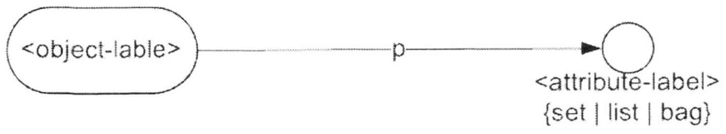


Figure 6.19: Constructional constraints (set, bag and list)

In the case of union, it is defined over a set of edges, as shown in Fig. 6.20.

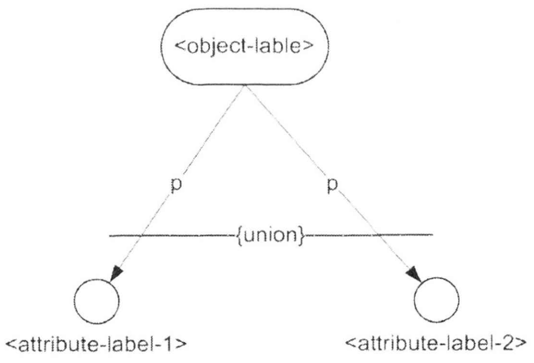


Figure 6.20: Constructional constraints (union)

Example 6.18: Fig. 6.8(b) shows an example of a set constraint defined for `goodsDescription`, where it is a *set* value with zero or more entries.

Example 6.19: Also, the `Goods-Sub-Type` has an attribute `Categories` that is of bag type with maximum of 5 entries, as shown in Fig. 6.21.

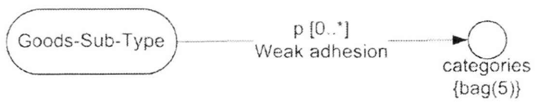


Figure 6.21: Constructional constraints (bag) example

Example 6.20: In the e-Sol, the conceptual view `Charge-Master-History` has an attribute of list type that stores the last 10 change histories for a given lot (`lotChargeAmount`), as shown in Fig. 6.22.

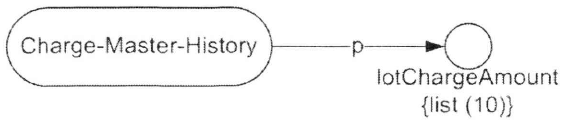


Figure 6.22: Constructional constraints (list) example

Example 6.21: In the e-Sol, the `Lot-Movement-History` date may be specified using typical date format or using week numbers, as shown in Fig. 6.23.

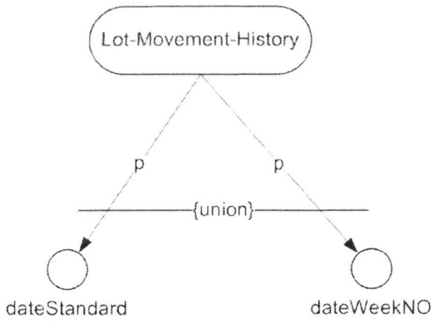


Figure 6.23: Constructional constraints (union) example

6.5 Schemata Transformation of Conceptual Views to Logical Views

In this section we look at some of the transformation of conceptual views and their properties (represented in XSemantic nets) to logical views (i.e. XML Schema (XSD)). The transformation is mainly concerned with mapping nodes, edges and constraints to XSD element/attribute declaration and/or simple/complex type definition. The transformation rules used (and extended) here are based on rules defined and described in (Feng, Chang & Dillon 2003) for stored (i.e. non-view) objects and their properties. Here it is adopted for transforming conceptual views.

The transformation of XSemantic net nodes to XSD considers:

(i) Depth: The transformation of XSemantic net to XSD is done using depth-by-depth (Feng, Chang & Dillon 2002) of the XSemantic node hierarchy, starting from the highest node depth.

(ii) Node type and content:

a. Basic nodes (atomic content) are mapped to XSD element or attribute types using XSD built-in atomic types, such as integer, sting, date, etc.

b. Complex nodes (and the corresponding hierarchy, relationships, constraints, etc.) are mapped to XSD by defining new (or extending existing) simpleType/complexTypes.

Here we outline only the constructs that require unique or extended transformation formalism to address the concept view properties.

6.5.1 Schemata Transformation of Nodes

6.5.1.1 Transformation of Basic Nodes

As stated earlier, a basic node in an XSemantic net may have atomic value or constructional value based on a collection of other basic types.

(i) Basic nodes

Basic nodes are mapped to corresponding XSD built-in simple types. The constraints defined for basic nodes (e.g. domain, restriction, etc.) are mapped to XSD using the facets mechanism provided in XSD (Feng, Chang & Dillon 2002; W3C-XSD 2001). In order to transform basic nodes to XSD, there exist two mechanisms: (a) map basic nodes to XSD attribute declaration; or (a) map basic nodes to XSD element (simpleType) declaration.

Let *Ns1* , *Ns2* be two basic nodes (with the node name *Ns1* and *Ns2*) in an XSemantic net. The resulting code segments of these basic nodes (of types string and float respectively) to an XSD element is shown below.

```
<!-- additional nesting -->
    <xs:element name="Ns1" type="xs:string"/>
    <xs:element name="Ns2" type="xs:float"/>
<!-- additional nesting -->
```

Also, the transformation of basic nodes to XSD includes domain constraints associated with the basic nodes. Transformation of such domain constraints (discussed in detail below in Section 6.5.2) to XSD is done by employing the XSD "facet" mechanism, by deriving new simple types (that stratifies the domain constraints of the basic node) by restricting existing simple types. For example, let Ns3 be a basic node that allows values between 0.10% to 0.60 of float type only. This restricted basic node can be transformed XSD as shown below.

```
<!-- additional nesting -->
<xs:element name="Ns3">
    <xs:simpleType>
        <xs:restriction base="xs:float">
            <xs:minInclusive value="0.00"/>
            <xs:maxInclusive value="0.60"/>
        </xs:restriction>
    </xs:simpleType>
</xs:element>
<!-- additional nesting -->
```

(ii) Basic nodes with constructional values

Basic nodes with constructional values (e.g. set, list etc.) are mapped to corresponding XSD constructional construct types, such as XSD list type (`<xs:list>`). XSD has built-in list and union type but no set/bag concepts. Transformation and type declaration of these basic nodes with constructional values are described in Section 6.6.2 below.

6.5.1.2 Transformation of Complex Nodes

A complex node is an XSemantic net that corresponds to a set of connection clusters. Thus, transformation of such nodes to XSD results in declaration of XSD complexType with schema constructs used to describe attribute, element and sub-element connections. Such complex type declarations will be used to constrain and validate schema instances (i.e. XML documents). As stated earlier, a conceptual view is usually represented using a complex node in an XSemantic net and transformed to a corresponding XSD complex type declaration.

Let *Nc* be a generic complex node (with the node name *Nc*) in the XSemantic net. The resulting code segments of this complex node to XSD complexType is shown below.

```
<!-- additional nesting -->
    <xs:complexType name="NcType">
                <!-- additional nesting -->
                <!-- additional nesting -->
    </xs:complexType>
<!-- additional nesting -->
```

6.5.1.3 Transformation of Semantic Relationships

Transformation of semantic relationships in XSemantic nets (i.e. labeled edges between complex nodes) is usually done as part of XSD complex type declarations. Depending on the relationships, complex type declarations may vary with additional facets, elements and/or constraints that are declared to logically state the relationships between one ore more XSD complex types. Here, we first consider the OO semantic relationships (as described in (Feng, Chang & Dillon 2002; Feng, Chang & Dillon 2003)) and their transformation followed by the description of the transformation of the "view" relationship in detail.

(i) of-property relationships

The transformation of of-property relationship results in a declaration of a new XSD attribute or a simple element. Also, in Section 6.5.2, transformation of of-property relationships in discussed in further detail.

Example 6.22: The transformation of of-property relationships in `Income` (Fig. 6.2) is shown below, using the *attribute* declarations.

```
<!-- additinal nesting -->
<xs:complexType name="IncomeType">
        <xs:attribute name="staffID" type="xs:ID" use="required"/>
        <xs:attribute name="baseSalary" type="xs:float" use="required"/>
        <xs:attribute name="totalBenefits" type="xs:float"/>
        <xs:attribute name="totalDeductions" type="xs:float"/>
        <xs:attribute name="weekNo" type="xs:integer" use="required"/>
        <xs:attribute name="month" type="xs:string" use="required"/>
        <xs:attribute name="year" type="xs:integer" use="required"/>
        <xs:attribute name="netSalary" type="xs:float" use="required"/>
    </xs:complexType>
</xs:schema>
<!-- additinal nesting -->
```

Code Listing 6.1: Transformation of of-property relationship using attribute declaration (example)

Example 6.23: As in the same example above, the transformation of the of-property relationships in `Income` (Fig. 6.2) is shown below, using *simple type* definitions. This is given in Code Listing 6.2.

```
<!-- additinal nesting -->
<xs:complexType name="IncomeType">
    <xs:sequence>
        <xs:element name="staffID" type="xs:ID" nillable="false"/>
        <xs:element name="baseSalary" type="xs:float" nillable="false"/>
        <xs:element name="totalBenefits" type="xs:float"/>
        <xs:element name="totalDeductions" type="xs:float"/>
        <xs:element name="weekNo" type="xs:integer" nillable="false"/>
        <xs:element name="month" type="xs:string" nillable="false"/>
        <xs:element name="year" type="xs:integer" nillable="false"/>
        <xs:element name="netSalary" type="xs:float" nillable="false"/>
    </xs:sequence>
</xs:complexType>
<!-- additinal nesting -->
```

Code Listing 6.2: Transformation of of-property relationship using element declaration (example)

(ii) Aggregation relationships

The mapping of an aggregation hierarchy in the XSemantic net to an XSD is straightforward as shown in the examples below.

Example 6.24: There exists an aggregation relationship (Fig. 6.3) between the whole-object `Staff` and part-objects `Salary-Pkg` and `Benefit-Pkg`. Also `Family-Support` and `Executive-Support` are part-objects of the whole-object `Benefit-Pkg` in the aggregation hierarchy. This is shown in the Code Listing 6.3.

Furthermore, in Section 6.5.2, the transformations of homogenous, ordered compositions and exclusive disjunctions are discussed in further detail.

(iii) Association relationships

In XSemantic nets, the transformation of association relationships to XSD can be achieved in two ways that are dependent on the associated class types, i.e. basic or complex.

(a) Case 1:

If the associated object node is a basic node, or a node that has brief information of simple structure, such nodes can be mapped as supplements of the source object (Feng, Chang & Dillon 2002).

(b) Case 2

If the associated object node is a complex node, the transformation is as follows. Let `A`, `B` be the source and associated objects in an association relationship with cardinality [0..*].

- Step 1: Declare both source and associated nodes as individual complex types, that is, `AType` and `BType`.
- Step 2: Declare an attribute or a subelement (say `a1`) in the source to refer to its associated complex node using XSD type IDREF or IDREFS, depending on the cardinality constraints associated with the association relationship. That is, if the cardinality constraint is either [1..1] or [0..1], the XSD construct `<xsd:IDREF>` is used and for others `<xsd:IDREF>` is used.
- Step 3: In the associated complex type, declare a key (e.g. using XSD ID construct) that is key referable (say `b1`).

- If sub-element was used in the source object to refer to the associated object, XSD KEY/KEYREF mechanism is declared to enforce the reference (similar to reference constraints described below in Section 6.5.2.

```
<xs:complexType name="StaffType">
    <xs:sequence>
        <!-- additinal nesting -->
            <xs:element name="Salary-Pkg" type="Salary-PkgType"/>
            <xs:element name="Benefit-Pkg" type="Benefit-PkgType"/>
            <!-- additinal nesting -->
    </xs:sequence>
</xs:complexType>
<!-- additinal nesting -->


<xs:complexType name="Salary-PkgType">
        <!-- additinal nesting -->
            <!-- additinal nesting -->
</xs:complexType>
<!-- additinal nesting -->


<xs:complexType name="Benefit-PkgType">
    <xs:sequence>
        <xs:element name="Family-Support" type="Family-SupportType"
                minOccurs="0" maxOccurs="unbounded"/>
        <xs:element name="Executive-Support" type="Executive-SupportType"
                minOccurs="0"/>
    </xs:sequence>
</xs:complexType>
<!-- additinal nesting -->

<xs:complexType name="Family-SupportType">
    <xs:sequence>
            <!-- additinal nesting -->
            <!-- additinal nesting -->
    </xs:sequence>
</xs:complexType>
<!-- additinal nesting -->

<xs:complexType name="Executive-SupportType">
    <xs:sequence>
        <!-- additinal nesting -->
        <!-- additinal nesting -->
</xs:sequence>
</xs:complexType>
```

Code Listing 6.3: Transformation of aggregation relationship (example)

The XSD code segment (Code Listing 6.4) demonstrates the transformation of the above steps (using attribute declaration). The Code Listing 6.5 demonstrates the transformation of the above steps using element declaration.

Example 6.25: The relationship between conceptual views `Site-Manager` and `Warehouse-Manager` is association, as shown in Fig. 6.4. `Site-Manager` *manages* `Warehouse-Manager` managers. The example mapping is given in Code Listing 6.6.

Example 6.26: The relationship between conceptual views `Warehouse-Manager` and `Collaborative-Partner` is association, as shown in Fig. 6.4. `Warehouse-Manager` *deals-with* `Collaborative-Partner`. The example mapping is given in Code Listing 6.6.

(iv) Generalization relationships

An example of a generalization relationship is shown in Fig. 6.4. The transformation of conceptual view generalization hierarchy is similar to mapping domain generalization hierarchy and can be achieved using three methods, namely, by creating new types by: (a) extension, (b) restriction and (c) redefine (Feng, Chang & Dillon 2002; Feng, Chang & Dillon 2003).

```
<!-- additional nesting -->
<xs:complexType name="AType">
    <xs:sequence>
        <xs:element name="a-Element-1">
        <!-- additional nesting -->
            <xs:complexType/>
        </xs:element>
    </xs:sequence>
    <xs:attribute name="b1" type="xs:IDREFS" use="optional"/>
</xs:complexType>

<!-- additional nesting -->
<xs:complexType name="BType">
    <xs:sequence>
        <xs:element name="b-Element-1">
            <xs:complexType/>
        </xs:element>
    </xs:sequence>
    <xs:attribute name="b1" type="xs:ID" use="required"/>
</xs:complexType>
<!-- additional nesting -->
```

Code Listing 6.4: Transformation of association relationship (attribute based)

Example 6.27: The relationship between conceptual view `Logistics-Staff`, `Admin` and `Site-Manager` is generalization/specialization, as shown in Fig. 6.4.

Example 6.28: The relationship between conceptual view `Warehouse-Staff` and `Warehouse-Manager` is generalization/specialization, as shown in Fig. 6.4.

```
<!-- additional nesting -->
<xs:complexType name="AType">
    <xs:sequence>
        <xs:element name="a-Element-1">
        <!-- additional nesting -->
            <xs:complexType/>
        </xs:element>
        <xs:element name="b1" type="xs:string" minOccurs="0" maxOccurs="unbounded"/>
    </xs:sequence>
</xs:complexType>

<!-- additional nesting -->
<xs:complexType name="BType">
    <xs:sequence>
        <xs:element name="b-Element-1">
            <xs:complexType/>
        </xs:element>
    </xs:sequence>
    <xs:attribute name="b1" type="xs:ID" use="required"/>
</xs:complexType>
<!-- additional nesting -->

<xs:keyref name="ReferToB" refer="BType">
    <xs:selector xpath="AType"/>
    <xs:field xpath="b1"/>
</xs:keyref>

<xs:key name="BType-ID">
    <xs:selector xpath="BType"/>
    <xs:field xpath="@b1"/>
</xs:key>
<!-- additional nesting -->
```

Code Listing 6.5: Transformation of association relationship (element based)

```
    <!-- additional nesting -->
    <xs:element name="Warehouse-Admin">
        <xs:complexType>
            <xs:sequence>
                <xs:element name="Site-Manager" type="Site-ManagerType"
                            maxOccurs="unbounded"/>
                <xs:element name="Warehouse-Manager" type="Warehouse-ManagerType"
                            maxOccurs="unbounded"/>
                <xs:element name="Collaborative-Partner" minOccurs="0"
                            maxOccurs="unbounded">
                    <xs:complexType>
                        <xs:complexContent>
                            <!-- additional nesting -->
                            <xs:extension base="CollaborativePartnerType">
                                <xs:attribute name="Warehouse-Manager_staffID"
                                            type="xs:IDREF" use="required"/>
                                <!-- additional nesting -->
                            </xs:extension>
                        </xs:complexContent>
                    </xs:complexType>
                </xs:element>
            <!-- additional nesting -->
         </xs:sequence>
      </xs:complexType>
      <!-- additional nesting -->
      <xs:key name="Warehouse-Mgr_ID">
         <xs:selector xpath="Warehouse-Manager"/>
         <xs:field xpath="@staffID"/>
      </xs:key>
      <xs:keyref name="Warehouse-Mgr_ID_REF" refer="Warehouse-Mgr_ID">
         <xs:selector xpath="Site-Manager"/>
         <xs:field xpath="managers"/>
      </xs:keyref>
   </xs:element>
   <!-- additional nesting -->
   <xs:complexType name="CustomerType">
      <xs:attribute name="customerID" type="xs:ID" use="required"/>
      <!-- additional nesting -->
      <!-- additional nesting -->
   </xs:complexType>
   <!-- additional nesting -->
   <xs:complexType name="Warehouse-ManagerType">
      <xs:complexContent>
         <xs:extension base="StaffType">
            <xs:sequence>
               <xs:element name="managedBy" maxOccurs="unbounded"/>
               <!-- additional nesting -->
            </xs:sequence>
            <!-- additional nesting -->
         </xs:extension>
      </xs:complexContent>
   </xs:complexType>
   <!-- additional nesting -->
   <xs:complexType name="StaffType">
      <xs:attribute name="staffID" type="xs:ID" use="required"/>
      <!-- additional nesting -->
      <!-- additional nesting -->
   </xs:complexType>
   <!-- additional nesting -->
   <xs:complexType name="CollaborativePartnerType">
      <xs:complexContent>
         <xs:extension base="CustomerType"/>
      </xs:complexContent>
   </xs:complexType>
   <!-- additional nesting -->
   <xs:complexType name="Site-ManagerType">
      <xs:complexContent>
         <xs:extension base="StaffType"/>
      </xs:complexContent>
   </xs:complexType>
   <!-- additional nesting -->
```

Code Listing 6.6: e-Sol example, transformation of association relationships

```
<!-- additional nesting -->
<!-- additional nesting -->
<xs:element name="eSol-Staff-Profile">
    <xs:complexType>
        <xs:sequence>
            <xs:element name="Site-Manager" type="Site-ManagerType"/>
            <xs:element name="Warehouse-Manager" type="Warehouse-ManagerType"
                        maxOccurs="3"/>
            <xs:element name="Logistics-Staff" type="Logistics-StaffType"
                        maxOccurs="unbounded"/>
            <xs:element name="Admin-Staff" type="AdminType" maxOccurs="unbounded"/>
            <xs:element name="Warehouse-Staff" type="Warehouse-Staff"
                        maxOccurs="unbounded"/>
            <!-- additional nesting -->
        </xs:sequence>
    </xs:complexType>
</xs:element>
<xs:complexType name="StaffType">
    <xs:sequence>
        <xs:element name="ID">
            <xs:unique name="staffID">
                <xs:selector xpath="StaffType"/>
                <xs:field xpath="@staffID"/>
            </xs:unique>
        </xs:element>
        <xs:element name="firstName" type="xs:string"/>
        <xs:element name="lastName" type="xs:string"/>
        <xs:element name="DOB" type="xs:date"/>
        <xs:element name="role" type="xs:string"/>
        <xs:element name="Address" type="xs:string"/>
        <!-- additional nesting -->
    </xs:sequence>
    <xs:attribute name="staffID" type="xs:ID" use="required"/>
</xs:complexType>
<xs:complexType name="Logistics-StaffType">
    <xs:complexContent>
        <xs:extension base="StaffType">
            <xs:sequence>
                <xs:element name="serviceProvider"/>
                <xs:element name="shiftTime" type="xs:time"/>
                <xs:element name="ReportTo"/>
                <xs:element name="Logistics-Company"/>
            </xs:sequence>
        </xs:extension>
    </xs:complexContent>
</xs:complexType>
<xs:complexType name="AdminType">
    <xs:complexContent>
        <xs:extension base="StaffType">
            <xs:sequence>
                <xs:element name="SiteID"/>
                <xs:element name="officeNo" type="xs:string"/>
                <xs:element name="Duties" type="xs:string"/>
                <xs:element name="reportTo"/>
                <xs:element name="Warehouse"/>
                <!-- additional nesting -->
            </xs:sequence>
        </xs:extension>
    </xs:complexContent>
</xs:complexType>
<xs:complexType name="Site-ManagerType">
    <xs:complexContent>
        <xs:extension base="Logistics-StaffType">
```

```
            <xs:sequence>
                <xs:element name="AdminDuties">
                    <xs:complexType>
                        <xs:complexContent>
                            <xs:restriction base="AdminType">
                                <xs:sequence>
                                    <xs:element name="SiteID"/>
                                    <xs:element name="officeNo" type="xs:string"/>
                                    <xs:element name="Duties" type="xs:string"/>
                                    <xs:element name="reportTo"/>
                                    <xs:element name="Warehouse"
maxOccurs="unbounded"/>
                                </xs:sequence>
                            </xs:restriction>
                        </xs:complexContent>
                    </xs:complexType>
                </xs:element>
            </xs:sequence>
        </xs:extension>
    </xs:complexContent>
</xs:complexType>
<!-- additional nesting -->
<xs:complexType name="Warehouse-StaffType">
    <xs:complexContent>
        <xs:extension base="StaffType">
            <!-- additional nesting -->
            <xs:sequence>
                <xs:element name="reportTo">
                    <xs:complexType>
                        <xs:simpleContent>
                            <xs:extension base="xs:IDREF">
                                <xs:attribute name="responsibleWarehouse-Manager"
                                            type="xs:IDREF" use="required"/>
                            </xs:extensin>
                        </xs:simpleContent>
                    </xs:complexType>
                    <!-- additional nesting -->
                </xs:element>
                <xs:element name="warehouseID"/>
            </xs:sequence>
        </xs:extension>
    </xs:complexContent>
</xs:complexType>
<xs:complexType name="Warehouse-ManagerType">
    <xs:complexContent>
        <xs:extension base="Warehouse-StaffType">
            <xs:sequence>
                <xs:element name="managementReports" type="xs:anyURI"
                        maxOccurs="unbounded"/>
            </xs:sequence>
            <!-- additional nesting -->
            <xs:attribute name="responsibleSite-Manager" type="xs:IDREF"
use="required"/>
        </xs:extension>
    </xs:complexContent>
</xs:complexType>
<!-- additional nesting -->
```

Code Listing 6.7: e-Sol example, transformation of IS-A relationships

6.5.1.4 Conceptual Views

The schemata transformation of conceptual views is analogous to the transformation of complex nodes in an XSemantic net, as described in Section 6.5.1.2. However, the mapping of an individual conceptual view is similar to mapping it to a predefined XSD template and later extending it to add individual conceptual view properties. This is because, depending on the domain in question (e.g. data warehouse model, simple web pages, etc.) and its requirements (e.g. remote schema location vs. customized local schema, etc.), the conceptual views may have, in addition to conceptual semantics and properties, additional XSD schema level semantics (e.g. view url, location, associated stylesheets, etc.).

For the conceptual views specified in using an XSemantic net, the transformation of such a conceptual view to a logical view can be conducted as follows:

- ***Step 1:*** First, by applying the transformation described for the complex node, in XSD, an abstract complex type definition (e.g. `<xs:viewType>`) is declared as shown in the Code Listing 6.7 below, for a simple such view (template) type. The abstract view type defined will form the XSD base type for further conceptual view transformation. This abstract type declaration may have additional semantics as we will show in Chapter 8 (in the context of dimensional conceptual views) and in Chapter 10 (in the context of website conceptual views). This type of transformation is similar to a template concept, where the basic structure of the logical view is defined and extended later on.
- ***Step 2:*** The conceptual views (for example, the conceptual view "Income" given in example 6.1) are transformed to XSD by using the above (step 1) viewType as the base (or template) and extending it (using the XSD extension mechanism) to transform other conceptual view properties, as shown in Code Listing 6.8.
- ***Step 3***: Other individual conceptual view properties (attributes, constraints, etc) and their relationships are mapped to the corresponding extended XSD type definitions (Code Listing 6.8).

The transformation described above is analogous to "deriving (complex) types by the extension concept" as described in (Binstock & al. 2002; Feng, Chang & Dillon 2002; Feng, Chang & Dillon 2003; W3C-XSD 2001).

```
<!-- additional nesting -->
    <xs:complexType name="viewType" abstract="true">
        <xs:sequence>
            <xs:element name="viewID" type="OIDType"/>
            <xs:element name="viewName" type="xs:string"/>
            <xs:element name="viewQuery" type="xs:anyURI"/>
            <xs:element name="viewLocation" type="xs:anyURI"/>
            <!-- further deployment specific properties -->
        </xs:sequence>
    </xs:complexType>
<!-- additional nesting -->
```

Code Listing 6.8: A simple logical view template (or abstract) structure

```
<!-- additional nesting -->
    <xs:complexType name="IncomeType">
        <xs:complexContent>
            <xs:extension base="viewType">
                <xs:sequence>
                    <xs:element name="baseSalary">
                        <xs:simpleType>
                            <xs:restriction base="xs:float">
                                <xs:minInclusive value="0.00"/>
                            </xs:restriction>
                        </xs:simpleType>
                    </xs:element>
                    <xs:element name="totalBenefits"/>
                    <!-- additional nesting -->
                </xs:sequence>
            </xs:extension>
        </xs:complexContent>
    </xs:complexType>
<!-- additional nesting -->
```

Code Listing 6.9: An example logical view code segment of the resulting transformation of the conceptual view "Income" as described in Example 6.1.

6.5.2 Schemata Transformation of Conceptual View Constraints

In the pervious section, we described the transformation of conceptual view (structural) properties such as attributes, relationships etc. In this section, we look at transformation of another important character that is unique to conceptual views, the conceptual view constraints.

6.5.2.1 Uniqueness Constraint

In order to transform uniqueness constraints to XML Schema, that is, to indicate that a particular element value is unique, we use the XML Schema "unique" construct. The unique construct first selects a set of elements and then identifies the element field relative to each selected element that has to be unique within the scope of the set of selected elements (Feng, Chang & Dillon 2002). It should be noted that uniqueness constraints may be applied to both element and attribute types. A general form of a transformed uniqueness constraint is given below:

```
<unique name= "<unique-constraint-name>">
       <selector xpath="<element-name" />
       <field xpath="<attribute-or-element-name>" />
</unique>
```

6.5.2.2 OID Constraint

In addition to mapping uniqueness constraints to XML Schema, to map OID to view schema, there are three options, namely (i) our own transformation; (ii) generic transformation methodologies proposed in such works as (Pardede, Rahayu & Taniar 2005a) to enforce unique constraints; and (iii) the new W3C recommendation of `<xml:ID>` (W3C-xml:ID 2005).

(i) Option 1

Our proposal to map the OID constraint to XML schema is to use XML Schema "unique" element in combination with the XSD "key" construct. The XSD unique element assures only the uniqueness when a key or data *exists*. This is different from the XSD "key" constructs where an entry must always exist. Thus, we use both constructs to map the conceptual view OIDs. For example, in XSD unique and key constructs are declared, as shown below:

```
<unique name= "<unique-constraint-name>">
       <selector xpath="<element-name" />
       <field xpath="<attribute-or-element-name>" />
</unique>

<key name= "<key-name>">
       <selector xpath="<element-name" />
       <field xpath="<attribute-or-element-name>" />
</key>
```

Therefore, to map conceptual view OID to XML Schema, we create a type called OIDType as shown below.

```
<!-- additional nesting -->
   <xs:complexType name="OIDType">
      <xs:sequence>
         <xs:element name="ID">
            <xs:unique name="OID">
               <xs:selector xpath="OIDType"/>
               <xs:field xpath="ID"/>
            </xs:unique>
         </xs:element>
      </xs:sequence>
   </xs:complexType>
<!-- additional nesting -->
```

Example 6.29: The uniqueness constraint shown in Fig. 6.15, (e.g. the `<<OID>>` in `Staff`) can be mapped to the view schema as shown in the code fragment (Code Listing 6.10) below.

```
<!-- additional nesting -->
  <xs:complexType name="staffType">
      <xs:sequence>
            <xs:element name="staffID" type="OIDType"/>
            <!-- additional nesting -->
            <xs:element name="managedBy">
               <xs:key name="manager">
                  <xs:selector/>
                  <xs:field xpath="staffID"/>
            </xs:key>
         </xs:element>
      </xs:sequence>
   </xs:complexType>
   <!-- additional nesting -->
   <!-- additional nesting -->
   <xs:complexType name="OIDType">
      <xs:sequence>
         <xs:element name="ID">
            <xs:unique name="OID">
               <xs:selector xpath="OIDType"/>
               <xs:field xpath="ID"/>
            </xs:unique>
         </xs:element>
      </xs:sequence>
   </xs:complexType>
<!-- additional nesting -->
```

Code Listing 6.10: Transformation of uniqueness constraint to logical view

(ii) Option 2

Another option is to map the OID like uniqueness constraints to XML Schema as was proposed by Pardede et al. (Pardede, Rahayu & Taniar 2005a, 2005b). They proposed to map uniqueness constraints from UML like modelling languages to XML Schema. A detailed discussion of their work can be found in (Pardede, Rahayu & Taniar 2005a, 2005b).

(iii) Option 3

A detailed discussion of the new W3C recommendation that deals with IDs and uniqueness constraints in XML schema can be found in (W3C-xml:ID 2005). This is one of the new approaches to transform conceptual view OIDs to logical view elements.

6.5.2.3 Referential Constraints

In transforming conceptual view referential constraints to logical view, two alternative solutions exist. They are (a) using XSD ID/IDREF constructs and (b) XSD KEY/KEYREF constructs. One of the major advantages of using KEY/KEYREF is that, it enables one to specify scope within which the uniqueness applies and it allows one to create KEY (and KEYREF) from a combination of any (simple, attribute or complex) content. The XSD declaration of KEY/KEYREF is similar to that of the uniqueness constraints.

```
<key name= "key-name">
        <selector xpath="element-name-1" />
        <field xpath="attribute-or-element-name-1" />
</key>


<keyref name= "keyref-name"       refer="key-name" >
        <selector xpath="element-name-2" />
        <field xpath="attribute-or-element-name-1" />
</key>
```

Example 6.30: In e-Sol, `Collaborative-Partner` is managed by one assigned `Warehouse-Manager` and served by one or more `Warehouse-Staff`. These relationships are maintained by the referential constraints (attribute based) as shown in Fig 6.4. The resulting transformation of such constraints to XSD is given in Code Listing 6.11.

```
<!-- addiional nesting -->
<xs:element name="Collaborative-Partner" type="CollaborativePartnerType">
    <!-- referential integrity constraint example -->
    <!-- addiional nesting -->
    <xs:key name="Warehouse-ManagerID">
        <xs:selector xpath="Warehouse-ManagerType"/>
        <xs:field xpath="@staffID"/>
    </xs:key>
    <xs:keyref name="Warehouse-ManagerID_REF" refer="Warehouse-ManagerID">
        <xs:selector xpath="Collaborative-PartnerType"/>
        <xs:field xpath="@warehouseManagerID"/>
    </xs:keyref>
    <xs:key name="WarehouseStaffID">
        <xs:selector xpath="Warehouse-StaffType"/>
        <xs:field xpath="@staffID"/>
    </xs:key>
    <xs:keyref name="WarehouseStaffID_REF" refer="WarehouseStaffID">
        <xs:selector xpath="Collaborative-PartnerType"/>
        <xs:field xpath="@warehouseStaffID"/>
    </xs:keyref>
</xs:element>
<!-- addiional nesting -->
<xs:complexType name="CollaborativePartnerType">
    <xs:attribute name="warehouseManagerID" type="xs:IDREF" use="required"/>
    <xs:attribute name="warehouseStaffID" type="xs:IDREFS"/>
    <!-- addiional nesting -->
</xs:complexType>

<xs:complexType name="Warehouse-StaffType">
    <xs:attribute name="staffID" type="xs:ID" use="required"/>
    <!-- addiional nesting -->
</xs:complexType>

<xs:complexType name="Warehouse-ManagerType">
    <xs:attribute name="staffID" type="xs:ID" use="required"/>
    <!-- addiional nesting -->
</xs:complexType>
<!-- addiional nesting -->
```

Code Listing 6.11: Transformation of referential integrity constraints (attribute based)

6.5.2.4 Ordered Composition

An ordered composition is mapped to the logical view using the XSD "sequence" construct. The following XSD code fragment demonstrates this.

```
<xs:complexType name = "whole-complextype-name" >
        <xs:sequence>
                <!- Additional nesting ->
                <xs:element name="part-name1" type="part-type1" />
                <xs:element name="part-name2" type="part-type2" />
                ...............
        </xs:sequence>
</xs:complexType>
```

Example 6.31: The following code segment in Code Listing 6.12 illustrates an e-Sol ordered composition example, as described in Example 6.13, the relationship between the whole-object `Lot-Movement-History` and its part-objects (`Internal-Lot-Movement-History, External-Lot-Movement-History`).

```
<!-- additional nesting -->
<!-- additional nesting -->
    <xs:complexType name="Lot-Movement-HistoryType">
        <xs:sequence>
            <xs:element name="Int-Lot-Movement" minOccurs="0"
                            maxOccurs="unbounded"/>
            <xs:element name="Ext-Lot-Movement" minOccurs="0"
                            maxOccurs="unbounded"/>
        </xs:sequence>
        <!-- additional nesting -->
    </xs:complexType>
    <!-- additional nesting -->
    <xs:complexType name="Int-Lot-Movement-HistoryType">
        <!-- additional nesting -->
    </xs:complexType>
<!-- additional nesting -->
<!-- additional nesting -->
    <xs:complexType name="Ext-Lot-Movement-HistoryType">
        <xs:sequence>
            <xs:element name="date"/>
            <!-- additional nesting -->
        </xs:sequence>
    </xs:complexType>
<!-- additional nesting -->
```

Code Listing 6.12: Transformation of ordered constraint to logical view (example)

6.5.2.5 Exclusive Disjunction

The conceptual view exclusive disjunction is mapped to logical views using the XSD "choice" constructor. For example, the following code fragment demonstrates this.

```
<xs:complexType name = "complextype-name" >
        <xs:choice>
                <!- Additional nesting ->
                ................
        </xs:choice>
</xs:complexType>
```

Example 6.32: As shown in Example 6.10, the exclusive disjunction constraint (Fig. 6.4) between `Internal-Lot-Movement` and `External-Lot-Movement` can be mapped XML Schema using the `<xs:choice>` schema construct, as shown in Code Listing 6.13.

```
<!-- additional nesting -->
<xs:element name="Lot-Movement">
<xs:complexType>
          <xs:complexContent>
          <xs:extension base="LotMovementType">
         <!-- additional nesting -->
          <xs:choice>
                <xs:element name = "Internal-Lot-Movement"
        type="InternalLotMovementType"/>
                <!-- additional nesting -->
                <xs:element name= "External-Lot-Movement"
            type="ExternalLotMovementType"/>
                <!-- additional nesting -->
          </xs:choice>
            <!-- additional nesting -->
          </xs:extension>
    </xs:complexContent>
</xs:complexType>
</xs:element>
<!-- additional nesting -->
```

Code Listing 6.13: Transformation of exclusive disjunction (example)

6.5.2.6 Constructional Constraints

The transformation of the constructional constraints to logical view schema is straightforward in the case of list and union types, while transformation of others such as set and bag needs to be derived from other types at the moment, as XML Schema does not provide native support for these types.

(i) List

XML Schema has a built-in list types (IDREFS, NMTOKENS etc.) which is a sequence of atomic types. Also, one can also map derived (or user-defined) list types to derived list types by using the XSD "list" constructs. In addition, XSD offers few facet mechanisms to refine the list constructs, namely (Feng, Chang & Dillon 2002; Feng, Chang & Dillon 2003); (a) length, (b) minLength, (c) maxLength and (d) enumeration. A simple XSD list declaration is shown below.

```
<xs:simpleType name="list-name" >
       <xs:list itemType = "XDS-base-types" />
</xs:simpleType>
```

Example 6.33: In the e-Sol, the `Goods-Sub-Type` has an attribute which is `Categories` that may contain 1 to 5 `Sub-Categories`. Code Listing 6.14 demonstrates this.

```
<!-- additional nesting -->
    <xs:simpleType name="Sub-Categories">
        <xs:list itemType="xs:string"/>
    </xs:simpleType>
    <!-- additional nesting -->

    <!-- additional nesting -->
    <xs:simpleType name="Categories">
        <xs:restriction base="Sub-Categories">
            <xs:length value="5"/>
        </xs:restriction>
    </xs:simpleType>
<!-- additional nesting -->
```

Code Listing 6.14: Transformation of constructional constraints – List (example)

(ii) Union

XML Schema provides support for the union constructional constraint using the "union" construct.

```
<xs:simpleType name="union-data-name" >
        <xs:union memberTypes= "xs-base-type" />
</xs:simpleType>
```

Example 6.34: In the e-Sol, the `Internal-Lot-Movement` date may be specified using a typical date format or using the week number of the year. The code fragment in Code Listing 6.15 illustrates this.

```
<!-- additional nesting -->
    <xs:simpleType name="Lot-Movement-DateType">
        <xs:union memberTypes="xs:date xs:integer"/>
    </xs:simpleType>
<!-- additional nesting -->
```

Code Listing 6.15: Transformation of constructional constraints – Union (example)

6.6 Conceptual View Dynamic Properties

The OO models are capable of describing both static and dynamic properties of a domain in question. In this section, we present a modelling notation to capture dynamic XML view properties in the layered view model, using XSemantic nets. It should be *noted* that the intention of this section is neither to propose a visual query builder for XML nor extensions to visual modelling of XML query language expressions. Also, as stated in Chapter 2, it is different from works such as (Augurusa et al. 2003; Braga, Campi & Ceri 2004, 2005; Oliboni & Tanca 2000, 2002), where a graphical representation is used to build and transform XML queries to query languages such as XPath and/or XQuery.

Our work here is two-fold: (a) it is focused on providing visual representation for the proposed *conceptual operators* (in comparisons to syntax specific query expressions) to construct conceptual views; and (b) provide modelling notations (specifically in XSemantic nets) to capture view dynamic properties such as generic methods and user defined methods.

To date, existing view models provide discussion only on the static properties of views and provide no mechanisms for capturing and/or modelling dynamic view properties. In a related work in Active XML (Abiteboul et al. 1999; Abiteboul et al. 2002a, 2002b), authors have provided extensive support for the dynamic properties for views in Active XML views, but active views are based on active rules rather than on data or document-centric view definitions.

In an XSemantic net, an *event node* is a node that describes a dynamic property (i.e. conceptual operators, methods, messages, or triggers) associated with a complex node, using one more conceptual operators, user defined and/or generic methods (i.e. get, set, update or delete).

Example 6.35: For example, in Fig 6.2 above, we have shown how a simple join conceptual operator is represented using an event node. In Fig. 6.24, the same example (Income) is shown with some generic methods.

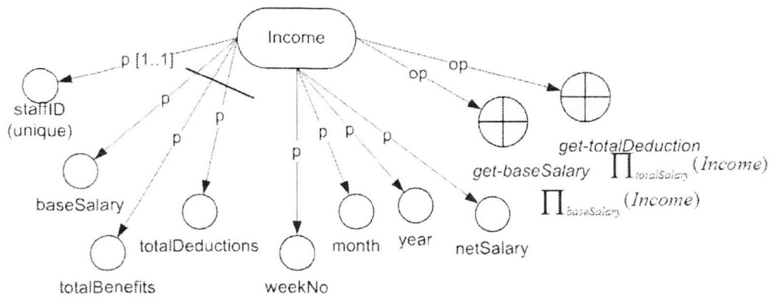


Figure 6.24: Conceptual view (Income) and generic methods

6.7 Declarative Transformation of Dynamic Properties into Document View Expressions

In this section, we discuss the declarative transformation of conceptual operators and dynamic view properties into document view query expressions. As stated in Chapters 4 and 5, such transformation is achieved using the transformation function given in equation 5.10.

Document view expressions are syntax specific query expressions (e.g. Chapter 3, equation 3.40) that are capable of querying native XML documents. There are few available language choices for document view expression. The following section discusses some of the available query language choices for XML (and document views), namely:

(i) W3C XPath (W3C-XPath 1999; W3C-XqFM 2005),

(ii) W3C XSLT (W3C-XSL 2003),

(iii) W3C XQuery (W3C-XqFM 2005; W3C-XQuery 2004; W3C-XQuery-UF 2006) and

(iv) non-query, procedural language such as Java or C++.

Also presented are the arguments for the selection of W3C XQuery as the document view query language in this research. Here, in the discussions and transformation of conceptual operators etc. to query languages, no assumptions are made in regards to: (a) XML storage structure, (b) query engine and (c) query (query language) performance. But it is assumed that the stored documents are valid XML documents and that the chosen query language is native to XML and supports XML data.

6.7.1 Choice of Document View Query Languages

6.7.1.1 XPath

XPath is a navigation based language that uses selection paths. The primary intended aim for the proposal of XPath is to provide a common syntax and semantics for functionality shared between XSL Transformations (XSLT) and XPointer (W3C-XPath 1999).

The primary purpose of this language is to address parts of an XML document. Starting from the root node of an XML document, XPath uses axes along which a section can occur. An axis denotes the connection between the originating node(s) and the node that is selected. An axis can be either a forward axis or a reverse axis.

XPath, unlike traditional query languages, due to its nature, does not provide the complete set of query language features and language syntax and readability is difficult when one needs to perform complex query operations such as join etc. Therefore, if XPath is selected as the document view language, it will provide minimal or no support for document view construction (support for conceptual operators) and no support for views and/or view data manipulation. Also, XPath does not always produce well-formed outputs and it has to be done externally to the query. Thus, XPath is not suited as the document view query language.

6.7.1.2 XSLT

XML Stylesheet Language & Transformation (XSLT) is a language for transforming XML documents into other XML documents, mainly intended for formatting. Unlike other currently available XML query languages, XSLT is a template based transformation language with support for iteration. It is mainly designed to use as part of XSL, which is a stylesheet language for XML. XSL specifies the styling of an

XML document by using XSLT to describe how the document is transformed into another XML document that uses the formatting vocabulary and this helps develop a particular presentation of the information.

In addition to XSLT, XSL includes an XML vocabulary for specifying formatting. Because of this, the language is difficult to read and understand. It can be used independently of XSL, but it is not intended as an XML transformation language (W3C-XSL 2003). Rather, it is designed primarily for the kinds of transformations that are needed when XSLT is used as part of XSL. Also, since it is presentation oriented. Only limited support is available for traditional query like features and, though it uses XPath for navigation, it is very difficult to accomplish querying of native XML documents, in a manner that is of use as a query language. Thus, XSLT is not well suited as a language for document views.

6.7.1.3 Non-query procedural language (e.g. Java, C++, etc.)

Querying native XML documents using custom procedures that are written in languages such as Java etc. is one of the best alternatives for querying and transforming XML documents. Since they are custom designed using commonly approved program structures such as DOM (W3C-DOM 2004) and/or SAX, they produce best performance and flexible query language features as they are written in procedural (non-query) languages.

But such approaches are non-standardized and do not allow uniformity for querying XML documents. Usually, the users need to familiarize themselves with various non-standard procedures in order to execute their queries and/or tasks. Also, in such approaches usually the query syntax tends to vary between different deployments platforms and query engines, even though they deal with similar XML documents. These approaches are well-suited for data-intensive applications (e.g data mining, ontologies), where performance of queries (and query modules) are more important than standards and/or query syntax. Therefore, though well suited for document views, to keep in line with the emerging XML query standards (to be widely accepted and deployed) and development, this approach is not used for document views, except there exists a requirement that cannot be done with the existing query language features and/or standards. For example, in Chapter 9, in the context of ontology views, this approach is adopted for the document view creating.

6.7.1.4 XQuery

XQuery is a relatively new XML query language proposal that is being adopted as the standard for querying XML documents. It is emerging as the replacement for relational SQL, as it is more functional and powerful than a simple ad-hoc query language. XQuery is a functional language (Chamberlin & Katz 2003) and its current functionalities (W3C-XQuery-UF 2006) are comparable to SQL standards (in the relational model).

In comparison with other XML query languages, it is more readable and incorporates XPath features as a subset. Although it still does not provide support for view definition and construction, it nevertheless provides user-defined function definitions, conditions, iterations and schema support for XML instances. Also, XQuery standards are still evolving and still being extended to support querying multiple data models, from relational, to XML and to Semantic Web (e.g. RDF, OWL) data models.

The formal semantics of XQuery can be found in (W3C-XqFM 2005) and the new working draft on XQuery update facilities in (W3C-XQuery-UF 2006). In this research, we briefly discuss XQuery as the document view language and for specifying view specific generic methods (namely the *get, set, delete* or *retrieve* methods). This is because XQuery standards do not fully support XML data manipulation yet. But, we choose XQuery as the document view constructor as it is gaining momentum as the language of choice for XML databases and repositories and in the future it will support many of the data manipulation features. Thus, XQuery is well-suited for our purpose as the document view query language better than other XML query languages (such as XPath or XSLT) as:

(i) XQuery is easy to read and write in comparison XPath and XSLT,

(ii) XQuery has powerful For-Let-Where-Order-Return (FLWOR) expression, in comaprions to XPath/XSLT are (purely) presentation oriented expressions,

(iii) XQuery provides User-Defined Functions (UDF) and variable binding and

(iv) XQuery support XML Schema.

(v) XQuery provide extension mechanism to support new functionalities and data models such as RDF, OWL, etc., such as in (Feng & Dillon 2005; Lehti & Fankhauser 2005)

6.7.2 Transformation of Conceptual Operators to Document View Expressions

In Chapters 3 and 4, a brief description of the transformation function that maps conceptual operators to query expressions (e.g. XQuery) was presented. Here, we consider this transformation in detail.

The transformation of conceptual operators to XQuery is a 2-step transformation: (a) the declarative transformation of conceptual operator definitions to W3C XQuery expressions; and (b) refinement and validation of XQuery expressions to query engine specific executable code (outside the scope of this research and not addressed in this research). The transformation is done in two steps so as:

(a) to keep the conceptual operators and textual XQuery expressions separated from the actual executable XQuery expressions (including both standalone and embedded code), to keep the transformation functions simpler (almost one-to-one declarative mapping between the conceptual operators to XQuery expressions) and independent of one specific query engine (and one predefined XQuery specification);

(b) to achieve MDA like PIM to PSM transformation with more emphasis on reducing vendor specific XQuery engine (or platform) specific syntax and maintain close proximity to the original W3C XQuery syntax;

(c) to support forthcoming (and new) XQuery standards (e.g. such as those proposed in (W3C-XQuery-UF 2006) and extensions (e.g. XQuery support for Semantic Web meta languages, such as OWL);

(d) to achieve portability and cross-platform interoperability between various present and future implementations of XQuery engines.

In addition, here we provide the basic (or skeletal) transformation, that is, it is declarative. This is because:

(i) XQuery standards are still evolving and providing a definitive non-declarative transformation may restrict the utilization of new XQuery standards;

(ii) if the transformation is generic (i.e. only the resulting skeletal syntax is defined), thus, the document view construction is left as part of the deployment option, that is;

  (a) document view expression may be stored as predefined functions within a database management system (e.g. Oracle, Tamino, SQL Server, etc., such as stored procedures, triggers or user-defined functions. Also, a given operator may also be mapped to a generic XQuery function template (analogues to User Defined Functions (UDF) (Rahayu 2000) in relational and Object-Relational (O-R) models). However, given the evolving nature of the XQuery standards, it is desirable to have the mapping to be as simple as possible, an approach used in this research, UDF syntax, notation, and definition may vary from programmer to programmer and between query engines;

  (b) document view expression may be deployed as embedded code within a script enabled page or external program modules (e.g. Oracle PL/SQL);

  (c) document view expression may be made part of the customized XQuery extensions, such as in (Feng & Dillon 2005) or (Lehti & Fankhauser 2005), where there exists a need to support application specific or domain specific LVM view construction.

(iii) Since the transformations produce skeletal XQuery expression, it can be customized and/or optimized by allowing it be extended (or restricted) to add platform specific requirements and/or environment settings.

It should be noted that the intention of this section is not to introduce and elaborate on XQuery syntax and functions, but rather to address the declarative mapping of conceptual operators to XQuery functions. The transformation described below also includes some of the XQuery proposed XQuery extensions by the working draft in (W3C-XQuery-UF 2006). Also, in real-world scenario, conceptual operators used will a combination of one ore basic sets as described above and can be mapped to a sequence of XQuery expressions using the core mappings described in the following sections. Table 1 illustrates a brief summary the proposed transformation described here.

Table 6.2: Summary of the transformation between conceptual level and document level concepts

	Conceptual Level	**Document Level**
Scope	Context	XML document/(s) and the associated schema
Schema	object schemas	XML schema
Operator	conceptual operators	XQuery operators / expressions
Operands	One or more conceptual objects in a given context	One or more XML files, node sequences and/or XPath expressions (XML data segments) in a given domain
Results	imaginary class (conceptual view)	imaginary instance document

Since the XQuery standards are still evolving, there exists no one query engine that allows all the functionalities of the XQuery specification. Thus, the XQuery syntax and expressions given here are executed and tested using XQuery engine from:

(i) Quip[2] prototype for XQuery

(ii) XQBE[3] engine (Braga, Campi & Ceri 2005).

(iii) All the query expression grammar is tested and validated using the W3C XQuery grammar test page applet[4].

6.7.2.1 Transformation of Conceptual Binary Operators to XQuery

The transformation of the conceptual binary operators to XQuery is to map the conceptual operator to XQuery set operators. Let `node-1` and `node-2` be two node sequences. Here, the nodes can be from one document or from two documents.

In the case of union operator, XQuery provide two lexical forms, namely "|" and "`union`". Thus the mapping between union conceptual operator to XQuery expression is straight as shown below in Code Listing 6.16. The XQuery parse tree generated by the W3C grammar test page is given in Appendix 6A.1, Code Listing A6.16 for reference purpose only.

```
let $a := document ("node-1.xml")
let $b := document ("node-2.xml")
return <union-result> {$a | $b} </union-result>
```

Code Listing 6.16: Mapping of union conceptual operator to XQuery union operator syntax

Similar to the transformation of the union conceptual operator, XQuery has a built-in operator for difference operations. The difference conceptual operator can be mapped to XQuery "`except`" operator, as shown Code Listing 6.17. In the code listing `node-1` and `node-2` are two node sequences.

[2] A Prototype for processing XQuery queries, Ver 2.2.1.1; http://xml.coverpages.org/ni2001-07-06-b.html
[3] XQBE (XQuery By Example); http://dbgroup.elet.polimi.it/xquery/XQBE.html
[4] W3C XQuery 1.0 Grammar Test; http://www.w3.org/2003/05/applets/xqueryApplet.html

```
let $a := document ("node-1.xml")
let $b := document ("node-2.xml")
return <difference-result>{$a except $b}</difference-result>
```

Code Listing 6.17: Mapping difference conceptual operator to XQuery except operator

In the case of intersection operation, the XQuery intersect operator can be used to map conceptual intersection operator. This is shown in Code Listing 6.18.

```
let $a := document ("node-1.xml")
let $b := document ("node-2.xml")
return <intersect-result>
       {$a intersect $b}</intersect-result>
```

Code Listing 6.18: Transformation of an intersection conceptual operator to XQuery intersect operator

In the case of the Cartesian product operator, the result can be achieved by mapping two node sequences to any XQuery FLWOR expression, when more than one node sequence is bound to a `FOR` clause (no where clause). This is shown in Code Listing 6.19.

```
for  $a in document ("node-1.xml"),
   $b in document ("node-2.xml")
return <cartesian-product>{$a} </a> </cartesian-product>
```

Code Listing 6.19: Transformation of a Cartesian product conceptual operator to XQuery FLWOR expression

The join conceptual operator can be mapped to an XQuery expression with the join condition being mapped to the `where` clause of the XQuery as shown in Code Listing 6.20. If no join condition present (i.e. natural join), the transformation is analogous to the Cartesian product, as shown in Code Listing 6.19 above. As stated in Chapter 5, the join condition can be of type simple, complex or pattern.

```
for    $a in document("node-1.xml"),
       $b in document("node-2.xml")
where $a/ID = $b/ID
return <simple-JOIN-example>
             <d1>{$a}</d1><d2>{$b}</d2>
       </simple-JOIN-example>
```

Code Listing 6.20: Transformation of a (conditional) join conceptual operator to XQuery FLWOR expression

The Code Listing 6.21 shows the pattern join operator mapping.

```
for    $a in document("node-1.xml"),
       $b in document("node-2.xml")
where $a/ID = $b/ID    (: join-condition :)
       and
       $a//join-element-2/ID2 = $b/ID2
return <pattern-JOIN-example>
             <d1>{$a}</d1><d2>{$b}</d2>
       </pattern-JOIN-example>
```

Code Listing 6.21: Transformation of a simple (conditional) join conceptual operator to XQuery FLWOR expression

6.7.2.2 Transformation of Conceptual Unary Operators to XQuery

The transformation of unary operators can be mapped directly to an XQuery FLWOR expressions, except for the rename and restructure operators. For the transformation of rename and restructure operators, we use the proposed XQuery extensions in the W3C working draft (W3C-XQuery-UF 2006) to XQuery *rename* and *transform* operators.

The *projection* operator is mapped to a simple XQuery FLWOR expression, usually with one-to-one mapping. That is, a FLWOR expression and the projection operator has similar operational semantics at the relevant level to abstraction (projection operator at the conceptual level and XQuery at the document level).

- The simplest projection operator $\Pi_*(node-1)$, that is, project a collection of nodes is also the simplest XQuery operator as shown in Code Listing 6.22. Here, the query will return the complete collection.

- As part of the projection operator semantics the specification of element (or node) selection that will appear in the result. This can be achieved by:

  (i) specifying element and attributes using their name, in an XPath (W3C-XPath 1999) syntax, for example, node-1//element/

  (ii) specifying elements and nodes using wildcards without using their names. For example:

    (a) `//` operator : specify all attributes and all descendants of the parent node (i.e. the node to which the operator is applied). This is defined in XQuery using the axis notation (W3C-XqFM 2005), as; `/descendant-or-self::node()/`

    (b) * : to specify all the elements of the parent node

    (c) `@*` : to specify any attribute

- In addition, the projection operator may be mapped to an XQuery expression with order semantics, using `order` clause, as shown in Code Listing 6.23.

```
doc ("node-1.xml")
```

Code Listing 6.22: The transformation of the simplest projection operator to XQuery

```
for    $a document ("node-1.xml")
order by $a/element-1, $a/element-1
return <a> {$a/*
       } </a>
```

Code Listing 6.23: Transformation of conceptual projection operator to XQuery expression

Similar to the projection operator, a conceptual selection operator mapped XQuery FLWOR expressions with the selection condition mapped to the `WHERE` clause of the XQuery expression. This syntax is given in Code Listing 6.24. It should be noted here that complex, nested selection conditions are also possible (i.e. complex type selection condition).

```
for    $a document ("node-1.xml")
where $a//selection-condition = value-or-nested-query
[order by]
return <a> {$a/*
       } </a>
```

Code Listing 6.24: Transformation of conceptual selection operator to XQuery expression

The rename conceptual operator is mapped to the new XQuery update facility `rename` operator. The syntax is given in Code Listing 6.25 -6.26. It should be noted here that the rename operator can be also used with projection and selection operator, using the new extensions provided for the XQuery FLWOR expression to support the rename operator.

```
rename
       {fn:document ("node-1.xml")/old-element-name}
to "new-element-name"
```

Code Listing 6.25: Transformation conceptual rename operator to XQuery expression (renaming an element)

```
rename
       {fn:document ("node-1")/nodes[1]/element-old-value[1]}
to $value0in-variable
```

Code Listing 6.26: Transformation conceptual rename operator to XQuery expression (renaming a value using a variable)

The restructure conceptual operator can be mapped to XQuery using:

(i) Implicit mapping: Here, the restructure operator is mapped to a sequence of FLWOR expressions together with optional `order` and/or `where` clause;

(ii) Explicit mapping: Here, the restructure operator is mapped the new XQuery update facility `transform` operator to create new strcuure of an existing document/(s). The syntax is given in Code Listing 6.27.

```
for    $a document ("node-1.xml")
where $a//selection-condition = value-or-nested-query
order by $a/element-2  (: new ordered structure :)
return <new-structure-1>
       {$a//element-4/*
       <new-structure-2>{$a//element-3/*
              <new-structure-3> {$a//element-1/*
              }</new-structure-3>
    }</new-structure-2>
  }</new-structure-1>
```

Code Listing 6.27: Transformation of conceptual restructure operator to XQuery expression (option (i))

Example 6.36: Code Listing 6.28 illustrates a generic restructure operator mapped to document view expression using the XQuery `transform` operator.

```
for $a in doc ("node-1.xml")//nodes
return
       transform
              copy $trans-a := $a
              do delete {$trans-a/node-1}
              return $trans-a
```

Code Listing 6.28: Transformation of restructure operator to XQuery expression (option (ii)

The Table 6.3 summarizes the transformations described above.

6.7.3 Document Manipulation and Weak Encapsulation in the LVM Views

Since the introduction of views in relational DBMS in early '80s, there have been constant discussions and research directions on data manipulation in views (including view updates) to date; view updatability is well-studied and implemented for relational environments in the context of materialized views and data warehouses. But, the concept of data manipulation in views, to date, has very few approved standards in both relational and OO models. Also, this is still a vendor/platform specific task (e.g. Oracle™ DBMS, IBM™ DB2, MySQL™ or O_2 OODBMS) and lacks consistency.

Table 6.3 Summary of the transformation of conceptual operators to document views (XQuery)

Conceptual Operator	**XQuery Equivalent**
Union operator	Built-in operator (`union` or `|`)
Intersection operator	Built-in operator `intersect`
Difference operator	Built-in operator `except`
Cartesian Product operator	In any FLWOR expression, when more than one variable is bound to a `FOR` clause (no where clause)
Join operator	Regular FLWOR expressions with join condition/(s).
Projection operator	XQuery FLWOR expression. Any valid FLWOR expression with valid return clause with non-updatable functions.
Selection Operator	Any valid FLWOR expression, with selective/conditional clause or UDF functions
Rename operator	XQuery `rename` operator `rename {$a/old-name[n] to $newname / "new-name"`
Restructure operator	(i) XQuery `transform` operator (ii) Nested selection/projection operators

Our intention here is neither to address view updatability issues in the LVM nor to put forward a proposal for view updatability issues using XQuery. Our focus here is to enable conceptual modelling of a minimal set of dynamic view properties (e.g. generic methods) and the corresponding transformation of such properties to document view query expressions. This, we argue, would provide some degree of *accessibility* and the *encapsulation* concept to views in the LVM. Our aim is to provide both static and dynamic modelling facility for the views in the LVM, at a higher-level of abstraction and the automated transformation its properties to view schema and query expressions.

However, it should be noted here that, in practice, view update issues are strongly coupled with the underlying (XML) database management system and the query language (i.e. supported operators, storage model etc.) and many restrictions and constraints may result from this. For the purpose of this research, we discuss only generic methods and use the newly proposed, extended W3C XQuery update facility (W3C-XQuery-UF 2006), which is still a W3C working draft at the time of this thesis submission. Thus, given below are a given set of *declarative* conditions that have to be met in order to perform querying and data accessibility in the LVM views for XML. These conditions are based on the view type and its nature, as described in Chapter 4, namely, Derived Imaginary XML Document (DID), Constructed Imaginary Document (CID), Triggered Imaginary Document (TID).

We divide the XML view update issues into three categories, namely:

(1) update of *pure views*: LVM views that are constructed using stored XML documents only;

(2) update of *imaginary views*: LVM views that are constructed using other views documents only; and finally

(3) update of *mixed views*: LVM views that are constructed using a mix of both stored and other imaginary (views) documents.

In addition, as described in Chapter 4, we consider the three types of views that can be constructed in the LVM, namely:

(a) Derived Imaginary XML Document (DID)

(b) Constructed Imaginary Document (CID)

(c) Triggered Imaginary Document (TID)

Due to the nature of these view types (the way they are constructed, constraints etc.), the effects of a view update vary significantly from type (a) to (c).

- **Case 1**: Data manipulation of *pure view* of type *DID*: these types of views are fully updatable (insert, delete, rename) except on any of the computed/summarized tag/(s).

- **Case 2**: Data manipulation of *pure view* of type *CID*: This type of views are updatable (insert, delete, rename) except on any of the computed, summarized or join condition tag/(s).
- **Case 3**: Data manipulation of *pure view* of type *TID*: These types of views are not updatable at any given time. In general *TID* views are read-only, but it is possible to construct other views using *TID* views.
- **Case 4**: Data manipulation of *imaginary view* of type *DID*: These types of views are updatable (insert, delete, rename), if and only if, the resulting updates do not violate stored document consistency (structure, hierarchy etc.). As in pure views, no updates are possible on computed/summarized tag/(s) and on views constructed over existing *TID* type views.
- **Case 5**: Data manipulation of *imaginary view* of type *CID*: These types of views are updatable (insert, delete, rename), if and only if the resulting updates do not violate stored document consistency and/or any join conditions. As in pure views, no updates are possible on computed/summarized tag/(s) and on views constructed over existing *TID* type views.
- **Case 6**: Data manipulation of *imaginary view* of type *TID*: These view types are usually not updatable. In reality, constructing imaginary views of type *TID* is illogical unless the base view/(s) are materialized in nature.
- **Case 7**: Data manipulation of *mixed view* of types *DID, CID & TID*: These types of views are only *selectively* updatable; that is, updatable under very strict conditions. During an update, many restrictions (from Case 1 - 6 above) may apply and maintaining consistency (especially in cyclic view definition) is a complex, challenging task. From our point of view, we argue that these types of views should be left unchanged in content, due to increased overheads in maintaining consistency among stored/imaginary XML documents.

A summery of permitted generic operations for view types in the LVM is given in Table 6.4. In addition, given the declarative conditions stated above are satisfied for a given view type in the LVM, we can summarise the transformation of its generic methods to XQuery expressions as given in Table 6.5.

Table 6.4: Summary of the generic operations permitted in the LVM

Operation (or method) / View types	**Derived Imaginary Document (DID)**	**Constructed Imaginary Document (CID)**	**Triggered Imaginary Document (TID)**
Get (or retrieve) methods	yes	yes	yes
Set (or insert) methods	conditional	conditional	no
Update methods	yes	conditional	no
Delete methods	conditional	conditional	no

It should be note here that, generic methods that satisfy the conditions above may also be mapped to XQuery using elaborate UDFs in XQuery (Chamberlin & Katz 2003). An example of such UDF XQuery syntax is shown in Code Listing 6.29. However, here in this research, we do not use such transformation. Another similar transformation methodology, to map generic and user defined methods to SQL, in the context of Object-Relational paradigm can be found in (Rahayu 2000).

```
define  function  get-Warehouse-Staff-firstName  ($param-staffID)as
element ()*
{
  for $staffMem in document ("staff.xml")//StaffMember
  where some $staffID in $staffMem/@staffID satisfies
       ($staffID = $param-staffID)
       and
       ($staffMem/Work/@workGroup = "warehouse")
  return $staffMem/@firstName
}
```

Code Listing 6.29: An example transformation of a generic *get* method (getFirstName()) in Warehouse-Staff to an XQuery UDF

Table 6.5: Summary of the transformation of LVM generic methods to document view expressions (XQuery)

Generic Methods	Comments	XQuery expression

Get methods	Simple FLWOR expression (simple project operators for attribute or elements)	`doc ("node-1.xml")//node/*`
		
		or
		
		`doc ("node-1.xml")//node/@*`

Update methods	XQuery `replace` operator
	
	* Note: here only replacing element values are considered.
		`replace` **`value of`**
		`{fn:doc ("node-1.xml")//element-a[1]`
		`with $new-value`

Set methods	XQuery `insert` operator
	
	* Note: here the new items are always assumed to be inserted as last
		`insert {$new-sub-node}`
		`as last into`
		`        fn: doc("node-1.xml")//nodes`
		`        /node[OID=$param-oid]`

Delete methods	XQuery `delete` operator	`delete`
		`{`
		`fn: doc("node-1.xml")`
		`        //nodes[element=$del-value]`
		`}`

6.7.4 Example Transformation of Conceptual Operators to Document Views in the e-Sol

In this section, we illustrate some of the e-Sol examples.

Example 6.37: As shown in Fig. 6.25, the conceptual selection operator of the view `Warehouse-Staff` can be mapped to the document view construct as shown below in the Code Listing 6.30.

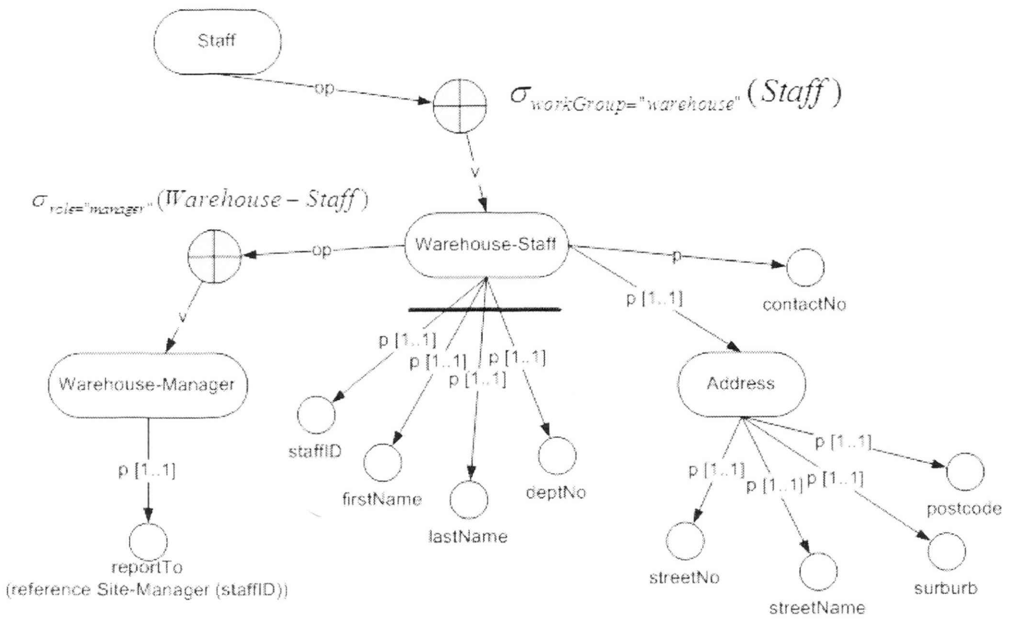


Figure 6.25: Conceptual view example (Warehouse-Staff)

```
for    $staff in document ("staff.xml")
where  $staff//StaffMember/Work/@workGroup = "warehouse"
order by $staff-member/lastName
return <Warehouse-Staff> {$staff/*
       } </Warehouse-Staff>
```

Code Listing 6.30: Transformation selection operator, $\sigma_{workGroup="warehouse"}(Staff)$ to XQuery FLWOR expression

Example 6.38: As shown in Fig. 6.25, the conceptual selection operator of the view `Warehouse-Manager` can be mapped to the document view construct as shown in Code Listing 6.31.

```
for    $staff in document ("staff.xml")
where  $staff//StaffMember/Role = "manager"
return <Warehouse-Manager> {$staff/*
       } </Warehouse-Manager>
```

Code Listing 6.31: Transformation selection operator, $\sigma_{Role="manager"}(Staff)$ to XQuery

Example 6.39: The Code Listing 6.32 illustrates an order (staffID, lastName, firstName, deptNo) constraint applied to the conceptual view Warehouse-Staff, as shown in Fig. 6.25.

```
for    $staff in document ("staff.xml")
where  $staff//StaffMember/Work/@workGroup = "warehouse"
order by $staff-member/lastName, $staff-member/firstName
return
       <Warehouse-Staff>
              <personal-info>
              (: ordered :)
                     <staffID> {$staff/@staffID} </staffID>
                     {$staff/lastName}
                     {$staff/@firstName}
                     <deptNo> {$staff/@deptNO} </deptNo>
              </personal-info>
              <Address> $staff/address/* </Address>
              <Contact-No> $staff/contactNo/* </Contact-No>
       </Warehouse-Staff>
```

Code Listing 6.32: Transformation of conceptual selection operator (with order constraint) to XQuery expression

Example 6.40: A Cartesian product conceptual operator (Fig. 6.26) can be mapped to the XQuery expression, at the document level as illustrated in Code Listing 6.33.

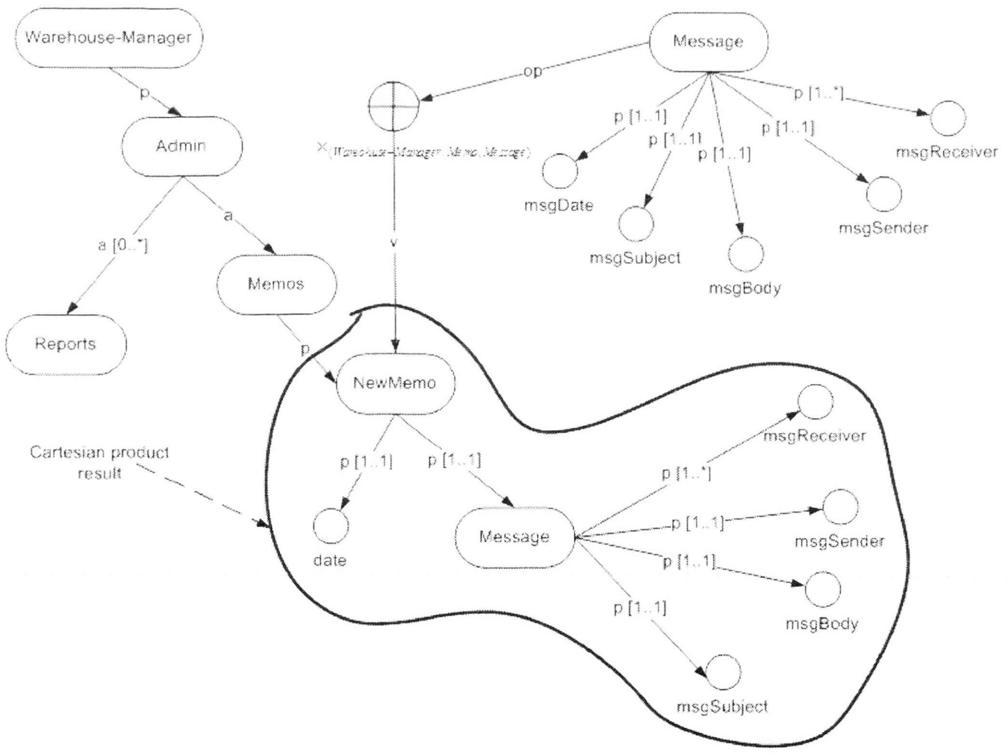


Figure 6. 26: Conceptual view construct (Cartesian product) example

```
for $memo in document ("Warehouse-Manager.xml")//Admin,
      $msg in document ("Messages.xml")//Msg
where $msg/@date= today()
return <Memo>  {
             $memo/Memos}
             <newMemo> {$msg/@*} {$msg/MsgBody} </newMemo>
       </Memo>
```

Code Listing 6.33: Transformation of $\times_{(Wareohuse-Manager\,/\,Memo,Message)}$ to XQuery expression (Cartesian product)

Example 6.41: The Code Listing 6.34 illustrates the transformation of the Income conceptual view construct (Fig. 6.2) to document view expression.

```
for $staff in document ("staff.xml")//Staff-member,
    $sal in document ("staff.xml")//Salary-Pkg,
    $benefits in document ("staff.xml")//Benefit-Pkg
let $totBenefits := $sal/Family-Support/totalAmount +
                    $sal/Executive-Support/totalAmount
let $netSal := $sal + $totBenefits - $sal/deductionAmount
where $staff/@staffID = $sal/@staffID and
         $staff/@staffID = $benefits/@staffID
return <Income> {$staff/@staffID}
  {$staff/FirstName}
  {$staff/LastName}
  {$staff/Tax-SSN}
  <baseSalary> {$sal/base} </baseSalary>
  <totalBenefits> {$totBenefits} </totalBenefits>
  <totalDeductions> {$sal/deductionAmount} </totalDeductions>
  <payMonth> {month (), year()} </payMonth>
  <netSalary > {$netSal} </netSalary>
</Income>
```

Code Listing 6.34: Transformation of the Income conceptual view construct $\rhd\lhd_{(x,y,z)} = (x \rightarrow_{(x.staffID=y.staffID)} y)\ and\ \ (x \rightarrow_{(x.staffID=z.staffID)} z)$ to XQuery expression

6.8 Conclusion

In this chapter we presented modelling and transformation methodology of conceptual views in the LVM using XSemantic nets. This chapter was logically grouped into two parts: first, the semantics of the XSemantic net; and second, the modelling and specifying of views in the LVM.

We first described and defined XSemantic net model elements and the proposed extensions (e.g. conceptual operators, methods, etc.) using formal semantics. Then we proceeded to illustrate modelling and specification of LVM conceptual views and its properties using the extended XSemantic nets using the e-Sol case study example. Later, we provided discussions of modelling and transforming conceptual view dynamic properties, namely the conceptual operators and the generic methods to document views, using illustrated examples. Here, in this research, we choose XQuery as the document view query language as its standard provides rich functionalities and quickly evolves to provide support for querying and manipulation in XML documents.

In Chapter 6, we presented another modelling and transformation methodology for conceptual views in the LVM using UML/OCL. Chapter 6 is supplementary to this chapter, where conceptual view semantics and properties are clearly defined and described using UML/OCL. In the chapters following Chapter 7, the modelling and transformation concepts used in the LVM (as discussed in Chapters 6 and 7) is utilized to develop architectural frameworks for two real-world application scenarios, namely, in modelling and transformation of dimensional data in an XML document warehouse framework (Chapter 8), view specification in the Semantic Web paradigm (Chapter 9) and website design in a view-driven, web engineering framework (Chapter 10).

References

Abiteboul, S, Amann, B, Cluet, S, Eyal, A, Mignet, L & Milo, T 1999, 'Active Views for Electronic Commerce', *Proceedings of the 25th International Conference on VLDB*, Edinburgh, Scotland, pp. 138-49.

Abiteboul, S, Benjelloun, O, Manolescu, I, Milo, T & Weber, R 2002a, 'Active XML: A Data-Centric Perspective on Web Services', *BDA*.

Abiteboul, S, Benjelloun, O, Manolescu, I, Milo, T & Weber, R 2002b, 'Active XML: Peer-to-Peer Data and Web Services Integration', *Proceedings of the 28th International Conference on VLDB*, HK, China.

Augurusa, E, Braga, D, Campi, A & Ceri, S 2003, 'Design and Implementation of a Graphical Interface to XQuery', *ACM Symposium on Applied Computing (SAC '03)*, ACM, Melbourne, USA, pp. 1163-7.

Binstock, C & al., e 2002, *The XML Schema Complete Reference*, Addison-Wesley, Boston.

Braga, D, Campi, A & Ceri, S 2004, 'XQBE: A Graphical Interface for XQuery Engines', *9th International Conference on Extending Database Technology (EBT '04)*, vol. LNCS 2992, Springer, Heraklion, Crete, Greece,, pp. 848-50.

Braga, D, Campi, A & Ceri, S 2005, 'XQBE (XQuery By Example): A visual interface to the standard XML query language', *ACM Transactions on Database Systems (TODS)*, vol. 30(2), no. 2, pp. 398-443

Chamberlin, DD & Katz, H 2003, *XQuery from the experts : a guide to the W3C XML query language*, Addison-Wesley, Boston.

Chen, YB, Ling, TW & Lee, ML 2002, 'Designing Valid XML Views', *Proceedings of the 21st International Conference on Conceptual Modeling (ER '02)*, vol. 2503, eds S Spaccapietra, ST March & Y Kambayashi, Springer-Verlag London, UK, Tampere, Finland, pp. 463 - 78.

Dillon, TS & Tan, PL 1993, *Object-Oriented Conceptual Modeling*, Prentice Hall, Australia.

Elmasri, R & Navathe, SB 2000, *Fundamentals of database systems*, 3 edn, Addison-Wesley, Reading, Mass. Harlow.

Feng, L, Chang, E & Dillon, TS 2002, 'A Semantic Network-based Design Methodology for XML Documents', *ACM Transactions on Information Systems (TOIS)*, vol. 20, No 4, no. 4, pp. 390 - 421

Feng, L, Chang, E & Dillon, TS 2003, 'Schemata Transformation of Object-Oriented Conceptual Models to XML', *International Journal of Computer Systems Science & Engineering*, vol. 18, No. 1, no. 1, pp. 45-60

Feng, L & Dillon, TS 2005, 'An XML-Enabled Data Mining Query Language XML-DMQL', *International Journal of Business Intelligence and Data Mining*, Lehti, P & Fankhauser, P 2005, 'SWQL – A Query Language for Data Integration Based on OWL', *First IFIP WG 2.12 & WG 12.4 International Workshop on Web Semantics (SWWS '05), In conjunction with On The Move Federated Conferences (OTM '05)*, Springer-Verlag, Agia Napa, Cyprus, pp. 936-46.

Oliboni, B & Tanca, L 2000, 'Querying XML Specified WWW Sites: Links and Recursion in XML-GL', *First International Conference on Computational Logic (CL '00)*, vol. LNCS 1861, Springer, London, UK, pp. 1167-81.

Oliboni, B & Tanca, L 2002, 'A visual language should be easy to use: a step forward for XML-GL', *Information Systems*, vol. 27(7), pp. 459-86

OMG-OCL 2003, *UML 2.0 OCL Final Adopted specification (http://www.omg.org/cgi-bin/doc?ptc/2003-10-14)*, OMG, 2005.

OMG-UML™ 2003a, *UML 2.0 Final Adopted Specification (http://www.uml.org/#UML2.0)*, OMG, 2005.

OMG-UML™ 2003b, *Unified Modeling Language™ (UML) Version 1.5 Specification*, OMG.

Pardede, E, Rahayu, JW & Taniar, D 2005a, 'Maintaining Data Consistency in XML-Based Applications', *3rd International IEEE Conference on Industrial Informatics (INDIN '05)*, IEEE Computer Society, Perth, Australia.

Pardede, E, Rahayu, JW & Taniar, D 2005b, 'Preserving Conceptual Constraints During XML Updates', *International Journal of Web Information Systems (IJWIS)*, vol. 1(2), no. 1, pp. 65-82

Rahayu, JW 2000, 'Object-Relational Transformation Methodology', Doctor of Philosophy (Ph.D) thesis, La Trobe University, Melbourne, Australia.

W3C-DOM 2004, *Document Object Model (DOM), (http://www.w3.org/DOM/)*, W3C, 2005.

W3C-xml:ID 2005, *xml:id Version 1.0 (http://www.w3.org/TR/2005/REC-xml-id-20050909/)*, The World Wide Web Consortium (W3C).

W3C-XPath 1999, *XML Path Language (XPath) Version 1.0 (http://www.w3.org/TR/xpath/)*, The World Wide Web Consortium (W3C), November 1999.

W3C-XqFM 2005, *XQuery 1.0 and XPath 2.0 Formal Semantics (http://www.w3.org/TR/xquery-semantics/)*, The World Wide Web Consortium (W3C), viewed November 2005.

W3C-XQuery 2004, *XQuery 1.0: An XML Query Language (http://www.w3c.org/)*, The World Wide Web Consortium (W3C), viewed November 2003 http://www.w3.org/TR/xquery.

W3C-XQuery-UF 2006, *XQuery Update Facility (http://www.w3.org/TR/2006/WD-xqupdate-20060127/) - W3C Working Draft 27 January 2006*, The World Wide Web Consortium (W3C), viewed January 2006.

W3C-XSD 2001, *XML Schema (http://www.w3.org/XML/Schema)*, W3C, 2004.

W3C-XSL 2003, *Extensible Stylesheet Language (XSL) (http://www.w3.org/Style/XSL)*.

Wouters, C 2006, 'A Formalization and Application of Ontology Extraction', Doctor of Philosophy (Ph.D) thesis, La Trobe University, Melbourne, Australia, Melbourne.

Chapter 7

Modelling and Transformation II

7.1 Introduction

In this chapter, we continue our discussion from Chapters 4 and 5 on the layered view model, similar to Chapter 6, where we focus on modelling and transforming conceptual views using OO concepts. Here, we focus on modelling and transformation of conceptual views using the OMG's Unified Modelling Language (UML) (OMG-UML™ 2003) to represent and specify conceptual semantics in the LVM. As in Chapter 6, here we extend the design methodology to model conceptual views (including both view data and behavior) using extension mechanisms provided by the UML.

In this chapter, our discussion here considers:

(i) enabling modelling *conceptual views* using UML (at the conceptual level) and

(ii) providing schemata transformation of conceptual views (represented in UML/OCL) to *logical view* schemas (at the logical level).

First, we describe our proposed notation to model conceptual views in UML. Secondly, we show how such modelling notations are transformed or mapped to view schema (XML Schema, at the logical model/schema level). For the schemata transformation of conceptual view model to XML Schema model, we adopt and extend the schemata transformation proposed by Feng et .al in (Feng, Chang & Dillon 2003; Xiaou et al. 2001a; Xiaou et al. 2001b).

It should be noted here that the modelling concepts presented in Chapters 4 and 5, and the description of modelling notions (e.g. classes, relationships, etc.), are used here without further description or elaboration. In this chapter we focus mainly on modelling and representation of the concepts using UML/OCL, as a continuation from Chapters 4-6.

The rest of the chapter is organized as follows. First, we provide a brief introduction to UML and OCL in section 7.2. Section 7.2 also includes a detailed discussion of modelling conceptual views using UML/OCL, the proposed stereotype extensions to UML to model conceptual view, its properties and relationships. In section 7.3, we provide a detailed discussion of the modelling of conceptual views and structural properties, followed by section 7.4 which presents modelling conceptual view constraints using UML/OCL. In section 7.5 we describe the schemata transformation of conceptual views (and their properties, constraints and relationships) specified using UML/OCL, using illustrated examples taken form the e-Sol case study described in Chapters 4 and 6. Section 7.6 concludes this chapter.

7.2 Modelling with UML/OCL

In this research, to demonstrate the generic view modelling and specification nature of conceptual views, we choose UML as our second modelling notation. We use UML as it is well understood by the wider development community and has established itself as the *de facto* software modelling language of choice. UML provides a well-defined rich collection of tools to model a given domain at the needed level of abstraction.

It can be said that UML helps to provide a well-defined blue print for a software system that is easily understood both by users and developers alike. UML also provides extensibility to the modelling language in the form of *stereotypes* which we utilise in defining our *conceptual views*. Another reason we adopt UML is that, as stated earlier, there already exist many schemata transformation rules to transform

UML models to XML Schema (Feng, Chang & Dillon 2003; Xiaou et al. 2001a; Xiaou et al. 2001b), that can be easily extendable to transform conceptual views to logical views.

As in Chapter 6, in this chapter we gradually build and illustrate our concepts using the e-Sol case study described in Chapter 4. A context diagram of the system, using the UML notation is given in Fig. 7.1. A detailed UML model of the e-Sol *domain* model is also given in Fig. 7.2 - 7.3, the model contents of the two logical groups in the e-Sol domain model, namely: the `Users` and Warehouse Management System (`WMS`) as shown in Fig 7.1 using the UML package diagram.

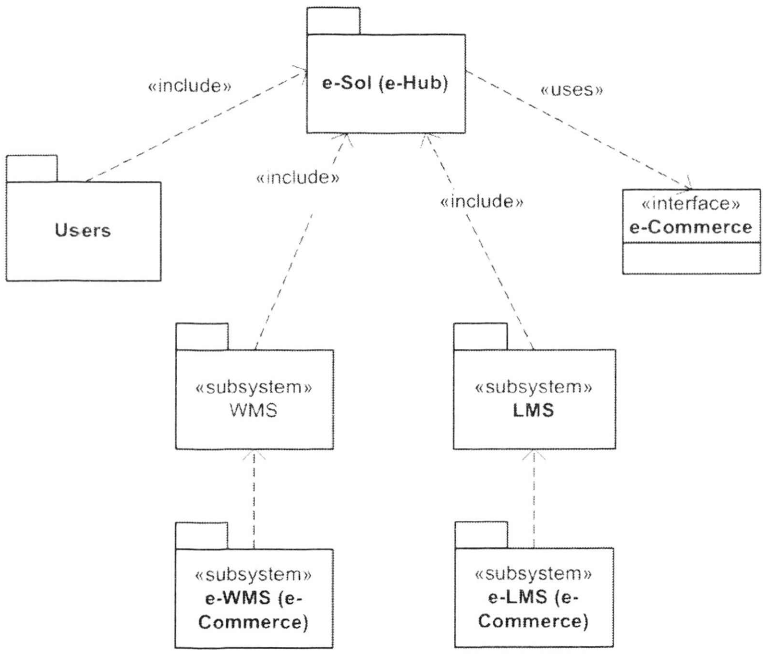


Figure 7.1: e-Sol system overview (context diagram)

It should be noted here that, in UML, the Object Constraint Language (OCL) (OMG-OCL 2003), which is now a part of the UML 2.0 standard, can support unambiguous constraints specifications for UML models including specification of static and dynamic (e.g. messages constraints) model elements. Here, in our conceptual view model, we incorporate OCL (in addition to built-in UML constraint specification features such as cardinality constraints (Dillon & Tan 1993; Feng, Chang & Dillon 2003), as our view constraint specification language to explicitly specify certain view constraints.

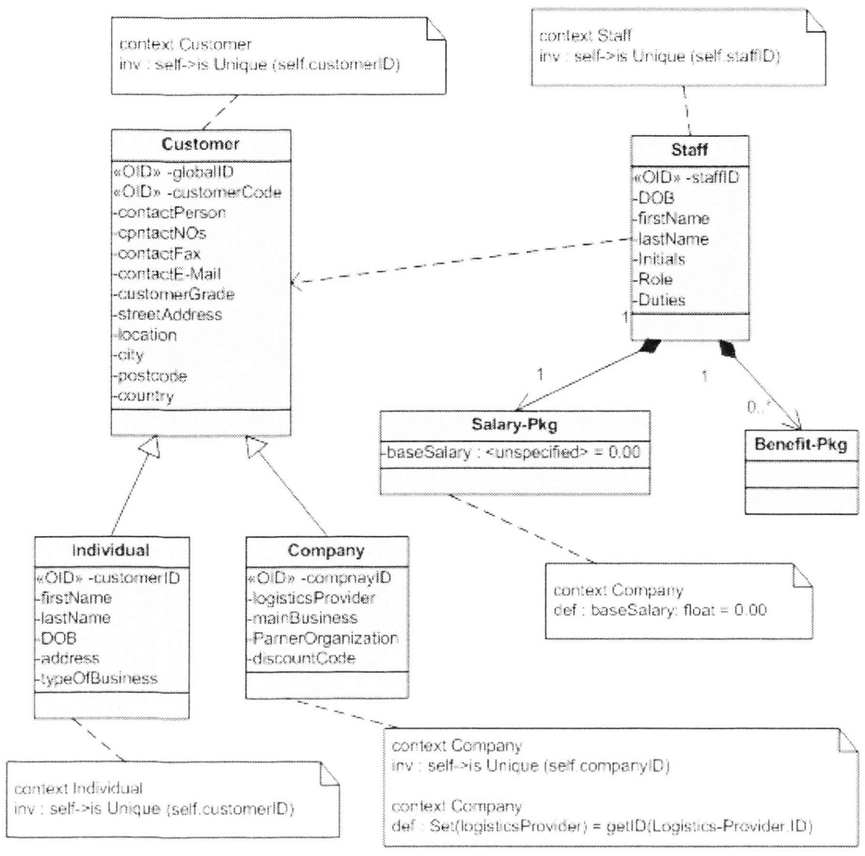


Figure 7.2: e-Sol users UML/OCL domain model (USER package)

7.3 Modelling with Conceptual Views using UML/OCL

To model conceptual views in UML, we introduce a set of *stereotypes* and visual constraint notations to represent conceptual views and their related properties (i.e. attributes, constraints, conceptual operators, etc.). In addition, to make view constraints more explicit and visible, as we have stated earlier, we use OCL. It should be noted here that the use of OCL in modelling conceptual views are supplementary to the UML visual model. We use OCL only to elaborate a conceptual view property further and not as a formal specification of the constraint itself. Also, it should be noted here that we do not use OCL to *define* or *specify* views, rather state additional constraints using OCL. This issue was discussed and argued for in Chapter 4.

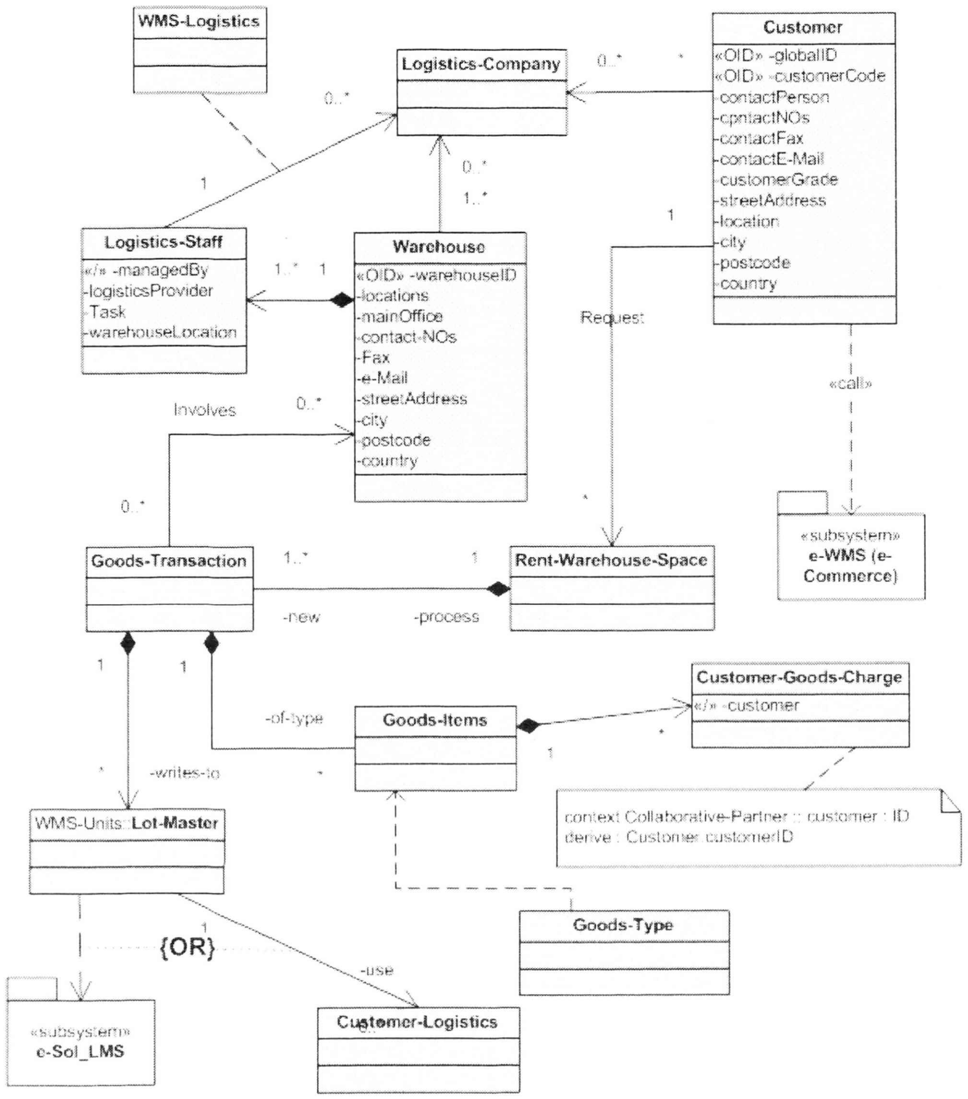


Figure 7.3: e-Sol simplified WMS UML/OCL domain model (WMS package)

However, OCL does support defining *derived* classes (OMG-OCL 2003; Warmer & Kleppe 2003), which is close to a view concept and some work such as in (Balsters 2003) utilize OCL for view specification. It is in the form of:

```
context:      <derived-class-name>::<new-attribute-name>: Type
derive:       <source-stored-class>.
        [some expression representing the derivation rule]
              / [<source-stored-class-attribute>
```

OCL also supports specifying derived values and attributes in already existing views (and stored classes) and specified in the form of:

```
context Typename::assocRoleName: Type
derive: -- some expression representing the derivation rule
```

However, as described in Chapters 2 and 4, this and works like (Balsters 2003) still concentrate on operational implementation, procedurally based definition of views albeit utilizing a higher level language such as OCL. This is not any easier to understand by the user than any other programming language.

7.3.1 Conceptual Views

Conceptual views are modelled in UML/OCL using the `<<view>>` stereotype. A *view* stereotype (Fig. 7.4) is a stereotyped class with conceptual view class representation, a set of view attributes (from one or more stored classes or derived via query constructs), a view constructor specification, optional view methods (e.g. derived attribute definitions etc.), and optional view constraints.

7.3.2 Conceptual Constructs

To show the relationship between a conceptual view and the stored class(es) from which it is constructed, we use a directed-dashed line with `<<construct>>` keyword shown above the line (Fig. 7.4). This is to avoid confusion with the built-in UML dependency relationship and other stereotypes.

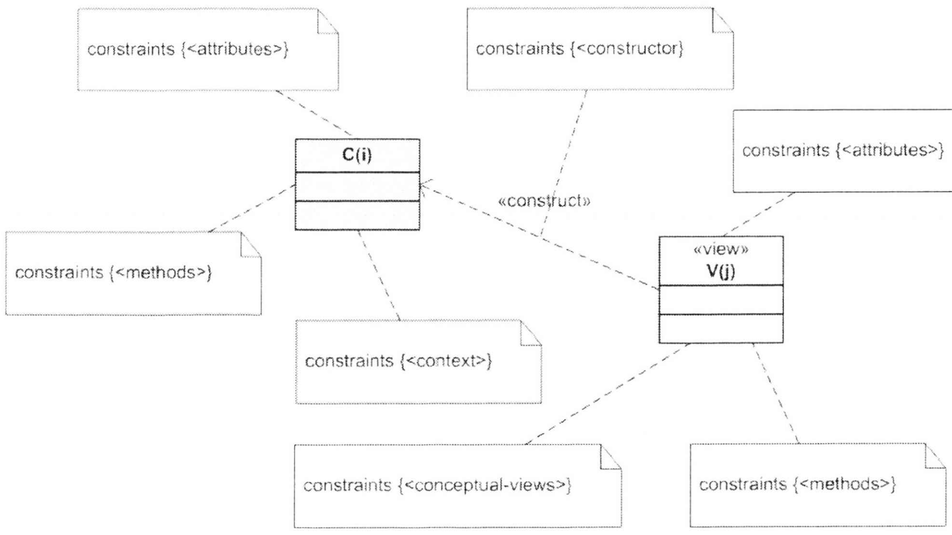

constraints {<attributes>}
constraints {<constructor}
C(i)
constraints {<attributes>}
«construct»
constraints {<methods>}
«view»
V(j)
constraints {<context>}
constraints {<conceptual-views>}
constraints {<methods>}

Figure 7.4: Conceptual view notation in UML/OCL

To model our conceptual views, we show view classes visually, with the `<<view>>` stereotypes and the relationship between the stored class and the view as `<<construct>>` stereotype. Therefore, we do not require non-visual OCL view specification as shown above, but may use it to show some of the derivation rules for the attributes and/or operations to make the view definition more explicit and precise. For example, as shown in Fig. 7.4, where a view V_j is constructed from a stored class C_i, the relationship is as `<<construct>>` relationship; the relationship that exists between a conceptual view and its stored class(es). If a conceptual view is constructed over an existing conceptual view (view of a view), the same relationship is used to show the hierarchy (the base conceptual view and the new conceptual view).

Example 7.1: In the e-Sol example, `Warehouse-Staff`, is a conceptual view in the context of `Staff`, as shown in Fig. 7.5 with conceptual operators, as shown below.

$$\texttt{Warehouse-Staff} = \sigma_{warehouse\text{-}Staff.Type=\text{“}warehouse\text{”}}(Staff)$$

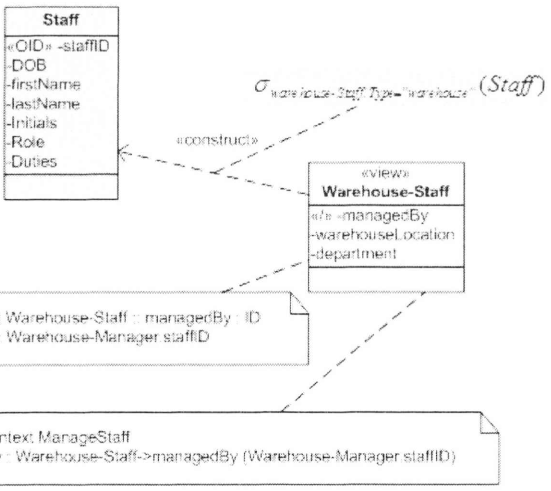

Staff
«OID» -staffID
-DOB
-firstName
-lastName
-Initials
-Role
-Duties
σ warehouse-Staff.Type="warehouse" (Staff)
«construct»
«view»
Warehouse-Staff
«/» -managedBy
-warehouseLocation
-department
context Warehouse-Staff :: managedBy : ID
derive : Warehouse-Manager.staffID
context ManageStaff
inv : Warehouse-Staff->managedBy (Warehouse-Manager.staffID)

Figure 7.5: e-Sol conceptual view examples (Warehouse-Staff, Warehouse-Manager)

Example 7.2: In the case of conceptual view `Income` (shown in Fig. 7.6 using UML/OCL), the conceptual construct is a conceptual JOIN operator with join conditions, where x = `Staff`, y = `Salary-Pkg` and z = `Benefit-Pkg`:

$$\texttt{Warehouse-Staff} = \begin{array}{l} \triangleright\triangleleft_{(x,y,z)} = (x \rightarrow_{(x.staffID=y.staffID)} y) \\ and \quad (x \rightarrow_{(x.staffID=z.staffID)} z) \end{array}$$

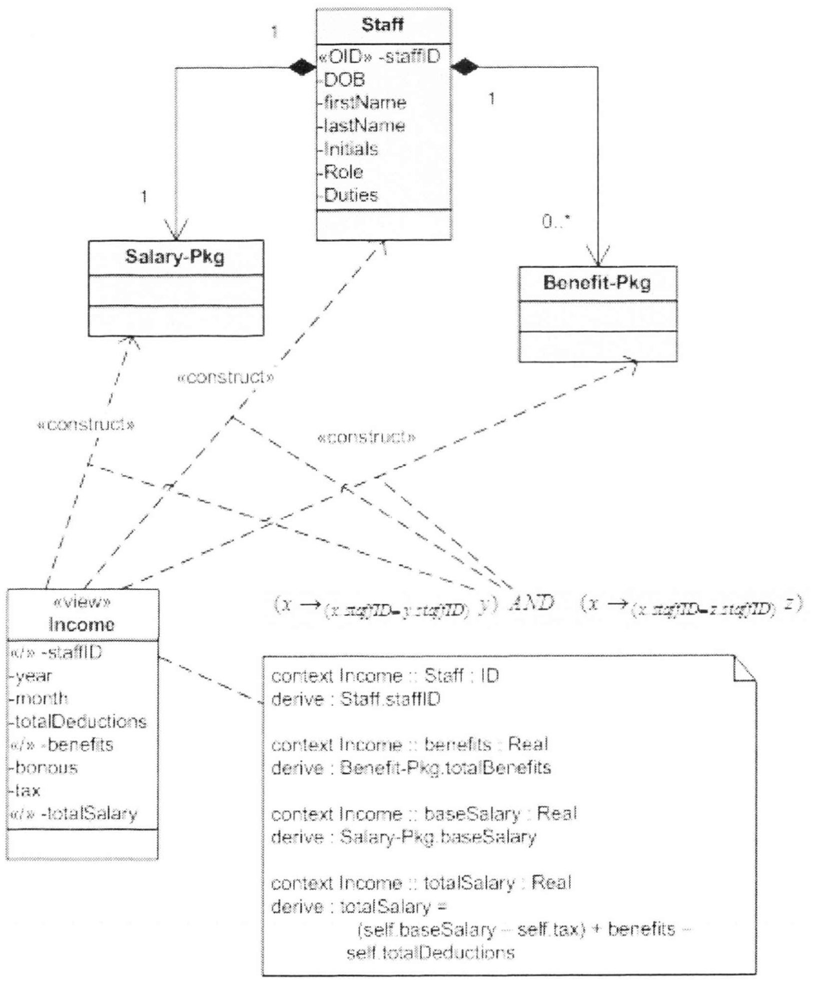
In addition, the following OCL statements hold true:

```
context Income :: Staff : ID
derive : Staff.staffID

context Income :: benefits : Real
derive : Benefit-Pkg.totalBenefits

context Income :: baseSalary : Real
derive : Salary-Pkg.baseSalary

context Income :: totalSalary : Real
derive : totalSalary = (self.baseSalary - self.tax) +
        benefits - self.totalDeductions
```



```
context Income :: Staff : ID
derive : Staff.staffID

context Income :: benefits : Real
derive : Benefit-Pkg.totalBenefits

context Income :: baseSalary : Real
derive : Salary-Pkg.baseSalary

context Income :: totalSalary : Real
derive : totalSalary =
        (self.baseSalary – self.tax) + benefits –
        self.totalDeductions
```

Figure 7.6: A conceptual view example in the e-Sol ("Income")

7.3.3 Conceptual View Relationships

In the case of conceptual view relationships such as generalization/specialization (i.e. IS-A), aggregation, association, the UML standard notations are used to represent them, as shown in the following examples.

Example 7.3: There exists an aggregation relationship (Fig. 7.7) between the whole-object `Staff` and part-objects `Salary-Pkg` and `Benefit-Pkg`.

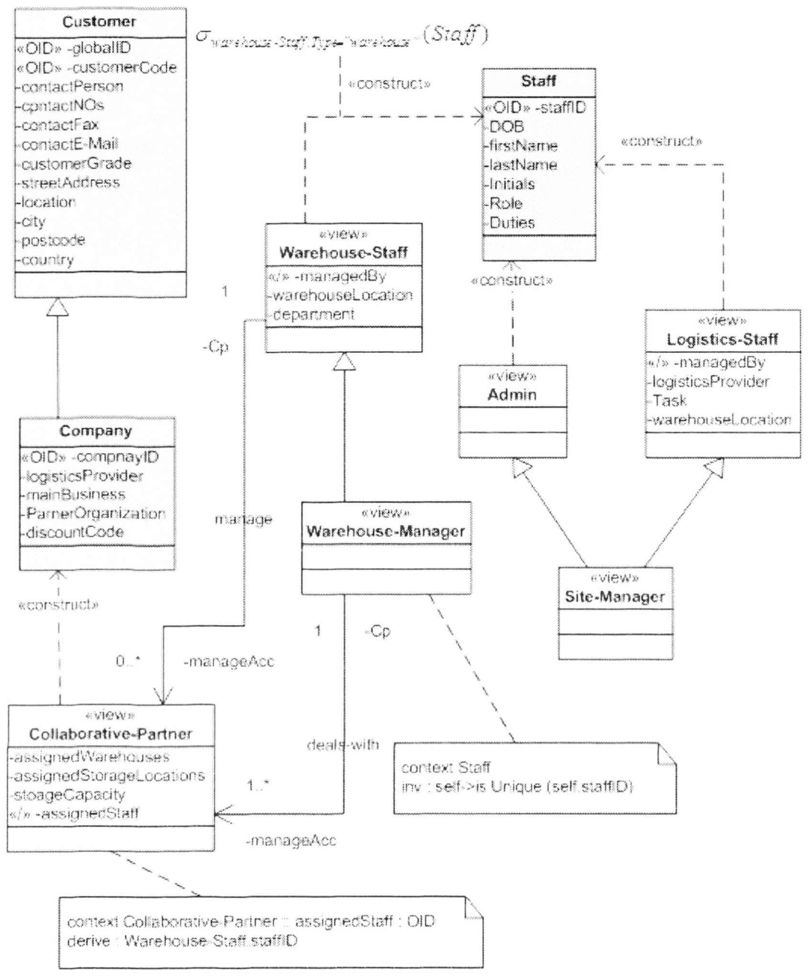


Figure 7.7: Conceptual view semantic relationship examples in the e-Sol

Example 7.4: The relationship between conceptual view `Logistics-Staff`, `Admin` and `Site-Manager` is generalization/specialization, as shown in Fig. 7.7.

`Site-Manager` *IS-A* `Logistics-Staff` and `Admin` type

Example 7.5: The relationship between conceptual views `Site-Manager` and `Warehouse-Manager` is association, as shown in Fig. 7.7.

`Site-Manager` ***manage*** `Warehouse-Manager` managers

Example 7.6: The relationship between conceptual views `Warehouse-Manager` and `Collaborative-Partner` is association, as shown in Fig. 7.7.

`Warehouse-Manager` *deals-with* `Collaborative-Partner.`

Example 7.7: The relationship between conceptual view `Warehouse-Staff` and `Warehouse-Manager` is generalization/specialization, as shown in Fig. 7.7.

Example 7.8: Also `Payment-History`, `WMS-Booking` and `LMS-Service` are part-objects of the whole-object `Customer-History` in the aggregation hierarchy, as shown in Fig. 7.8.

7.3.4 Logical Grouping of Conceptual Views

As we stated earlier, the role of conceptual views is to provide different perspectives to a stored document class hierarchy. In modelling complex systems (e.g. e-Sol) designers usually group semantically related conceptual model artefacts into one or more groups to reduce complexity and/or better understand, model and design a system in question. For example, in our e-Sol example, as shown in Fig. 7.1, the e-sol model is grouped into sub-models namely:

(i) *e-Hub:* the central system that represents the e-Sol and provides global semantics and connectivity to the other sub-systems;

(ii) *Users:* a logical grouping, semantics and representation of all the users and participants of the e-Sol;

(iii) *Warehouse Management Systems (WMS):* an e-Sol sub-system for local warehouses to do their warehouse booking, order processing, capacity management, etc.;

(iv) E*lectronic Warehouse Management Systems (e-WMS:* a web based e-sol sub-system for warehouse customers and their staff to manage and monitor their storage, inventory, booking etc.;

(v) *Logistics Management System (LMS):* e-Sol sub-system for local logistics service providers to perform, manage and track their business functions as part of the integrated e-Sol system;

(vi) an equivalent e-WMS system for logistics for the e-Sol customers; e-LMS. To book, track and manage their logistics.

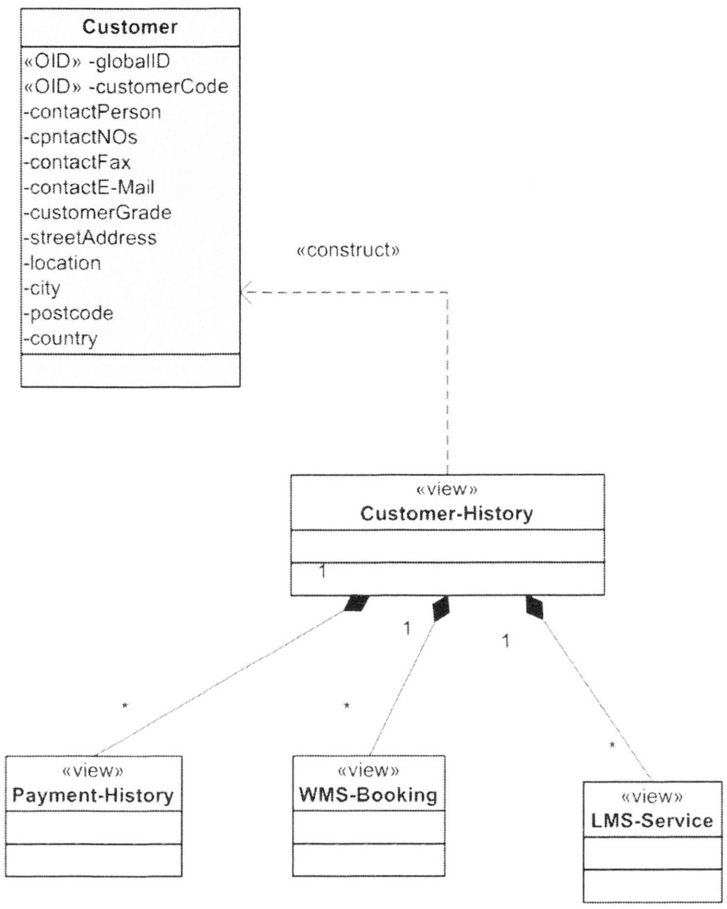


Figure 7.8: Aggregation relationships examples

These groupings or sub-systems are similar to that of a *subject area* (Coad & Yourdon 1991; Dillon & Tan 1993) (or class categories (Booch 1993)) in OO conceptual modelling techniques.

Similar to the above scenario, in the case where there are semantically connected or linked conceptual views and conceptual view hierarchies, they too can be grouped into logical groups/hierarchies. When we allow logical grouping of conceptual views and their associated relationships into a subject group, we are giving the designer the abstraction needed to model a cluster of conceptual views, without worrying about external connectivity of the view cluster.

Conversely, in order to capture this logical grouping in UML, it provides a modelling notation called ***package***. According to OMG specification (OMG-UML™ 2003);

> *"A package is a grouping of model elements. Packages themselves may be nested within other packages. A package may contain subordinate packages as well as other kinds of model elements. All kinds of UML model elements can be organized into packages".*

Thus, based on the above definition, it can be said that it describes our logical or subject grouping of the conceptual views into groups. Given a conceptual view, in order to include additional semantics/refinements, we can construct additional new view-hierarchies. These hierarchies may form additional structural or dependency relationship with existing conceptual views or view hierarchies.

To represent conceptual view groups, similar to the domain model notation, we use UML packages diagrams with `<<view>>` stereotype, as shown in Fig. 7.9.

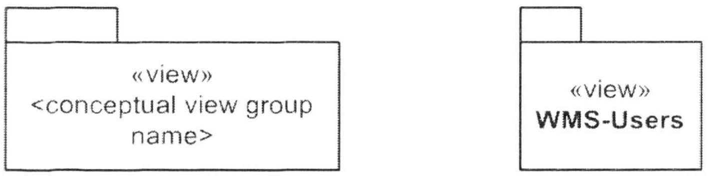


Figure 7.9: A conceptual view package notation and an example (WMS-Users package)

The application of package notation in domain modelling is not new. Some work has already been done in investigating packages for dimensional modelling (Sergio Lujan-Mora, Juan Trujillo & Song 2002a, 2002b). In our research, we utilize the conceptual view `<<view>>` packages for grouping of semantically related conceptual views in the LVM as one group. Some of the benefits and applications of such packages in the LVM (and their utilization) are described in Chapters 8 and 10, in the context of dimensional modelling and web engineering, respectively.

7.4 Conceptual View Constraints

This section provides a discussion of how conceptual view constraints are visually specified in UML/OCL. UML provide a rich set of notations to describe model constraints. We do not provide detailed discussion on the constraints themselves, as their formal semantics was given in Chapter 5 and were described in detail in Chapter 6. Here, we discuss their UML/OCL based modelling notations only.

Also, as stated in Chapter 6, unlike XSemantic nets, in UML constraints are not specific and semantically poor for specifying constraints for semi-structured data (Feng, Chang & Dillon 2002; Feng, Chang & Dillon 2003), as in the case of modelling views in the LVM. Also, some constraints in UML are not visually elaborate and additional (usually textual) descriptions are needed using declarative languages such as OCL. In the rest of the section, we provide a discussion of modelling constraints that are characteristic of conceptual views using the UML/OCL notation.

(i) Cardinality Constraint

As described in Chapter 6, the cardinality constraint shows the number of instances of one conceptual view that may relate to a single instance of another. In UML, to represent cardinality constraints we use the UML cardinality constraints that are part of the UML relationship notations. To describe class / attribute relationship cardinality (i.e. of-property relationship cardinality in XSemantic nets, as described in Chapter 6), we use OCL expressions.

(ii) Homogenous Composition

In an aggregation relationship, if part-object(s) is of same type as the whole-object, it is described as a homogenous composition. This can be shown as a typical aggregation relationship, with keyword "`homogenous`" stated in the labelled edge.

Example 7.9: In the e-Sol case study, `Executive-Pkg` may have additional benefit packages of similar types. This is shown in Fig. 7.10.

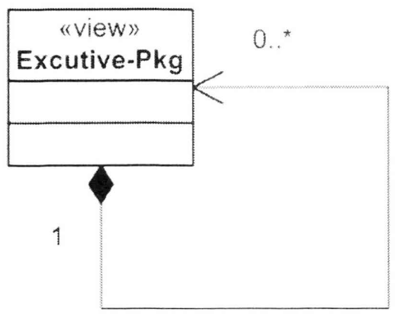


Figure 7.10: Homogenous composition example

(iii) Ordered Composition

To represent the ordered composition visually for conceptual views using UML/OCL, we add a numbered annotation (stereotype) that allows capturing of the ordered composition as shown in Fig. 7.11, utilizing stereotypes to specify the objects' order of occurrence such as <<1>>, <<2>>, <<3>>, ….,<<n>>.

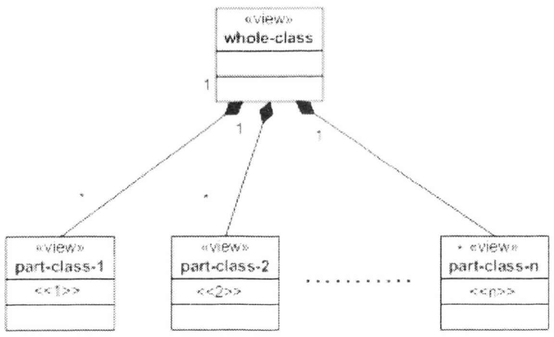


Figure 7.11: Ordered-Composition

Example 7.10: In the e-Sol. example, in the composition relationship between whole-object `Lot-Movement-History` and (`Internal-Lot-Movement-History, External-Lot-Movement-History`) is *ordered*. This is shown in Fig. 7.13.

(iv) Dependency / Adhesion Constraint

To represent adhesion (or dependency) constraints, we denote a textual keyword "`strong adhesion`" or "`weak adhesion`" over the UML dependency relationships as shown in Fig. 7.12.

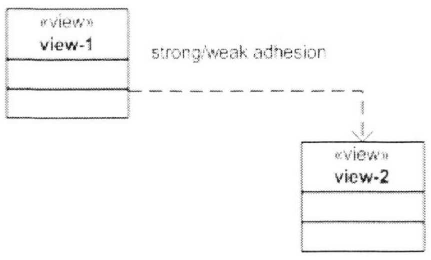


Figure 7.12: Adhesion constraint

Example 7.11: In the case of conceptual views `Lot-Movement-History`, `Internal-Lot-Movement-History` and `External-Lot-Movement-History`, the relationship has weak adhesion as shown in Fig. 7.13, as `Lot-Movement-History` may or may not have either internal or external lot movement histories.

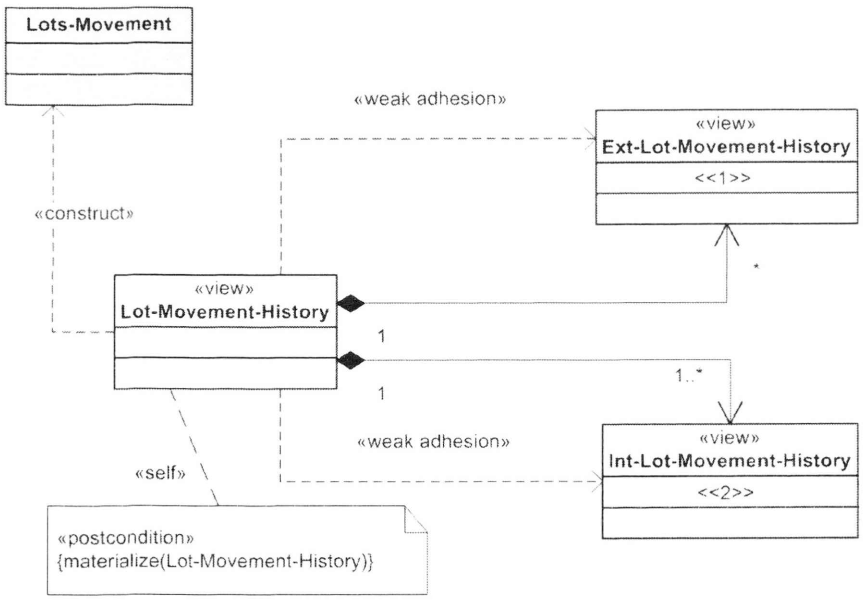


Figure 7. 13: Ordered and adhesion constraint examples in e-Sol

Example 7.12: In the case of conceptual view `Warehouse-Manager` the attribute `staffID` is compulsory and shown as strong adhesion in Fig. 7.7 together with the following OCL statement.

```
context Warehouse-Manager
inv : self->isRequired(staffID)
```

Example 7.13: In the case of conceptual views Warehouse-Manager and Warehouse-Staff, in the context of Staff (Fig. 7.7), we indicate the adhesion relationship between them using the following OCL statements given below.

```
context Warehouse-Staff :: managedBy : ID
derive : Warehouse-Manager.staffID

context Warehouse-Manager
inv: self.responsibleFor := Set(Warehouse-Staff.staffID)

context ManageStaff
inv : Warehouse-Staff->managedBy (Warehouse-Manager.staffID)
```

Example 7.14: Conversely, in the case of conceptual view `Goods-Sub-Type` the attribute `goodsDescription` is an optional *set* value with zero or more entries and shown as weak adhesion using the following OCL statement in Fig. 7.15.

```
context Goods-Sub-Type
inv : goodsDescription->nullAllowed = true
```

(v) Exclusive Disjunction

To visually represent exclusive disjunction constraint between two or more conceptual views, we denote the "OR" keyword between the object relationships as shown in Fig. 7.14. It should be noted that the exclusive disjunction is only allowed between association and aggregation relationships.

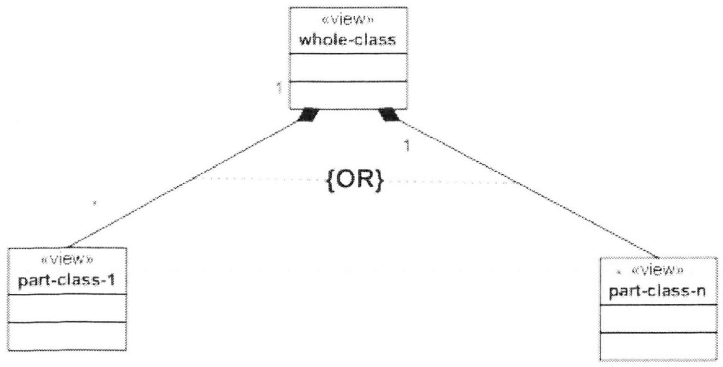


Figure 7.14: Exclusive disjunction

Example 7.15: In the case of conceptual views `Lot-Movement` (e-Sol), the exclusive disjunction between `Internal-Lot-Movement` (stored goods change owners) and

External-Lot-Movement (goods shipped outside the warehouse) can be represented as shown in Fig. 7.15.

Example 7.16: In the case of LMS (sub-system) and Customer-Logistics, exclusive disjunction can be shown using the OCL statement "OR" between the relationships (as shown in Fig. 7.15).

(vi) Uniqueness Constraint

As described in Chapters 5 and 6, to represent uniqueness constraints in UML/OCL, in addition to the UML model elements, additional OCL expressions are used as there exist no visual notation to represent uniqueness constraints. The OCL expression using the isUnique operation, as shown below;

```
context <contextual instance name>
inv : self->isUnique (<unique attribute name>)
```

(vii) OID Constraint

Conversely, to visually model OID constraint in conceptual views using UML class diagram, we define a new stereotype, OID as an attribute type with corresponding uniqueness constraint OCL expression, shown in Figs. 7.7 and 7.15. Together with attribute name and optional type definition, stereotype <<OID>> can be used in UML to indicate that the attribute that is a unique OID.

Example 7.17: In the case of conceptual view Logistics-Staff (Fig. 7.15), managedBy attribute is unique, as each logistics staff reports to only one site manager. But Logistics-Staff has a unique OID, staffID.

```
context Logistics-Staff
inv : self->isUnique(managedBy)
inv : self->isUnique(staffID)
```

Example 7.18: In the case of conceptual view Warehouse-Manager (Fig 7.7), we indicate the unique staffID by the following OCL expression;

```
context Staff
inv : self->isUnique(self.staffID)
```

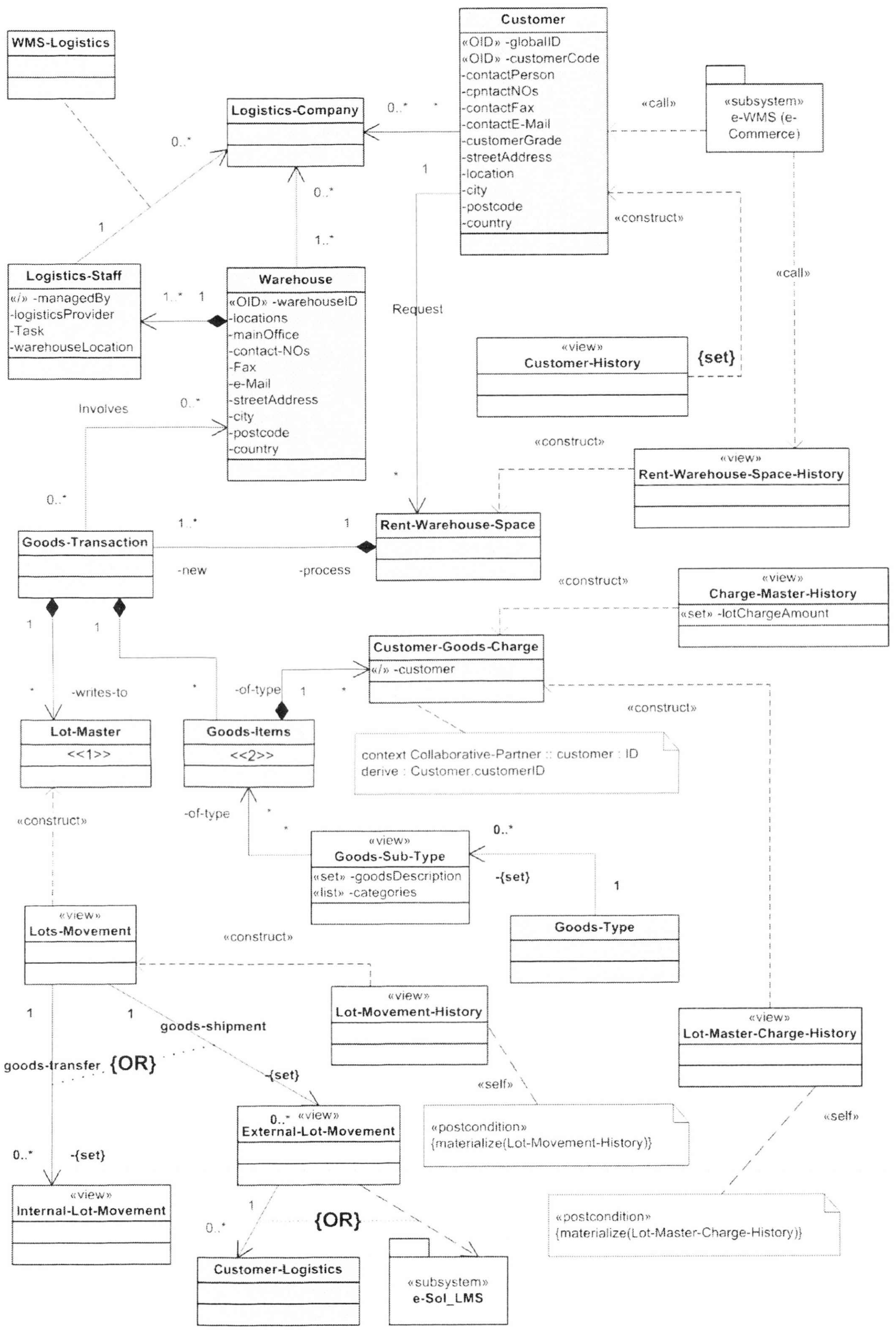


Figure 7.15: e-Sol domain objects and conceptual views (WMS)

(viii) Constructional Constraints

As described in Chapters 5 and 6, constructional constraints include set, bag, list and union. A union constructional constraint allows one type to be constructed using union of one or more atomic and list types. In Chapter 6, such constraints were illustrated using XSemantic nets. Here, we use UML and accompanying OCL expressions to describe these constraints.

To represent set, list and bag constraints for attributes in the conceptual views, we denote them visually using UML stereotypes, namely `<<set>>`, `<<list>>` and `<<bag>>` respectively, as shown in Fig. 7.16. Also, to further elaborate the constraints in detail (e.g. length, size, type, etc.), we use OCL expressions for each constructional constraint as illustrated in the examples below.

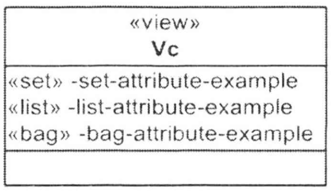

«view»
Vc
«set» -set-attribute-example
«list» -list-attribute-example
«bag» -bag-attribute-example

Figure 7.16: Constructional constraints

It should be noted that in OCL there exists Set, OrderedSet, Sequence and Bag constructs. An OCL OrderedSet is a set with items order and a sequence is a bag whose elements are ordered, that is, equivalent to a list concept.

```
context Vc
derive : self->Set(set-attribute-example)
derive: self->Sequence(list-attribute-example)
derive: self->Bag(set-attribute-example)
```

Example 7.19: In `Goods-Sub-Type` (Fig. 7.15) shows an example of a set constraint defined for `goodsDescription`, where it is a *set* value with zero or more entries.

```
context Goods-Sub-Type
derive: Set(goodsDescription)
inv: self.goodsDescription->size =>0
```

Example 7.20: In the e-Sol, the conceptual view `Charge-Master-History` has an attribute of list type that stores the last 10 change histories for a given lot (`lotChargeAmount`), as shown in Fig. 7.15.

Example 7.21: The `Goods-Sub-Type` attribute `Categories`, which allows maximum of 5 entries, may be stated using OCL (Fig. 7.15) as follows;

```
context Goods-Sub-Type
derive : Categories = Sequence(Sub-Categories)

context Goods-Sub-Type
inv: Categories->size <= 5
```

Example 7.22: In the e-Sol, the conceptual view Charge-Master-History has an attribute of list type that stores the last 10 change history for a given lot (`lotChargeAmount`), as shown in Fig. 7.15.

```
context Charge-Master-History
derive : self.lotChargeAmount := OrderedSet (charges)

context Charge-Master-History
derive : self.lotChargeAmount->maxLength()= 10
```

To define union constraints for conceptual view attributes, a non-visual OCL expression is used as shown in the following example.

Example 7.23: In the e-Sol, the date in the conceptual view, `Int-Lot-Movement-History` may be specified using (a) typical date format or (b) using week numbers. This is expressed in the following OCL expression.

```
context Internal-Lot-Movement
derive : self.Lot-Movement-Date := union (date, integer)
```

In the case of specifying constructional constraints to conceptual views, we use the notation presented in (Rahayu 2000), where the keywords set, list or bag is specified at the conceptual class representation as shown in Fig. 7.15. Also, OCL expressions may be used to express these constraints further as in the case of attributes.

Example 7.24: In the e-Sol, the conceptual views `Internal-Lot-Movement`, `External-Lot-Movement` are sets as shown in Fig. 7.15.

```
context Lot-Movement
inv: self->Set(Internal-Lot-Movement)
inv: self->Set(External-Lot-Movement)
```

7.5 Transformation of Conceptual Views To Logical Views

In this section we look at some of the transformation issues of conceptual views specified in UML/OCL to logical views. The transformation used here is based on (Feng, Chang & Dillon 2003; Xiaou et al. 2001a; Xiaou et al. 2001b) and extended here to map conceptual views and their properties to XML Schema (XSD). A summary of such transformations are given in Table 7.1.

7.5.1 Conceptual Views

Similar to the case of XSemantic nets (Chapter 6), the transformation of conceptual views represented in UML/OCL to XML Schema is similar to the schemata transformation of UML classes and the associated constraints, as described in (Feng, Chang & Dillon 2003). Also, similar to the case in XSematic nets, first the abstract `<<view>>` model is transformed to an abstract XSD complex type (or template) and later, each individual views type is declared using the "deriving (complex) types by extension concept" as described in (Binstock & al. 2002; Feng, Chang & Dillon 2002; Feng, Chang & Dillon 2003; W3C-XSD 2001).

The transformation of such conceptual views to logical views can be conducted as follows:

- *Step 1:* First, by applying the transformation described for a class an abstract XSD complex type definition (e.g. `<xs:viewType>`) is declared.
- *Step 2:* The conceptual view (e.g. `Warehouse-Staff`) is derived by extending the step 1 abstract `xs:ViewType` as shown in Code Listing 7.1.

- ***Step 3***: Other individual conceptual view properties (attributes, constraints, etc) and their relationships are mapped to the corresponding extended XSD type definitions (Code Listing 7.1).

```
<!-- additional nesting -->
    <xs:complexType name="ViewType" abstract="true">
        <xs:sequence>
            <xs:element name="viewID" type="OIDType"/>
            <xs:element name="viewName" type="xs:string"/>
            <xs:element name="viewQuery" type="xs:anyURI"/>
            <xs:element name="viewLocation" type="xs:anyURI"/>
            <!-- further deployment specific properties -->
        </xs:sequence>
    </xs:complexType>
    <!-- additional nesting -->
    <xs:complexType name="OIDType"/>
    <!-- additional nesting -->
    <!-- additional nesting -->
    <xs:complexType name="Warehouse-StaffType">
        <xs:complexContent>
            <xs:extension base="ViewType">
                <xs:sequence>
                    <xs:element name="firstName" type="xs:string"/>
                    <xs:element name="lastName" type="xs:string"/>
                    <xs:element name="DOB" type="xs:date"/>
                    <xs:element name="streetAddress" type="xs:string"/>
                    <xs:element name="suburb" type="xs:string"/>
                    <xs:element name="postcode" type="xs:string"/>
                </xs:sequence>
                <!-- additional nesting -->
                <xs:attribute name="staffID" type="xs:ID" use="required"/>
                <xs:attribute name="managedBy" type="xs:IDREF" use="required"/>
                <!-- additional nesting -->
            </xs:extension>
        </xs:complexContent>
    </xs:complexType>
<!-- additional nesting -->
```

Code Listing 7.1: An example transformation of conceptual views to logical view (Warehouse-Staff)

7.5.2 Conceptual View Relationships

The transformation of conceptual view relationships specified in UML/OCL are similar to the case in XSemantic nets as illustrated in Chapter 6. Here, we provide some worked out examples to demonstrate the transformation of UML relationships to XSD.

7.5.3 Conceptual View Constraints

Here, it should be noted that, in the LVM, the OCL expressions are used as a declarative textual notation to describe or elaborate conceptual view properties further and are *not* used to specify or declare view properties. Thus the transformation of OCL expression in LVM is declarative, that is each expression is mapped and not *evaluated*. Thus, the transformation of conceptual view constraints specified using UML/OCL to XML Schema is similar to that is described in Chapter 6, Section 6.5.2. The only difference is that the constraints are described using UML and OCL expressions but the resulting XML Schema syntax is exactly the same as the in the case of XSemantic nets.

Example 7.25: The unique constraint shown in n Fig. 7.7 (e.g. the `<<OID>>` in `Staff`) can be mapped to the view schema as shown in the code fragment (Code Listing 7.2) below.

```
<!-- additional nesting -->
   <xs:complexType name="staffType">
        <xs:sequence>
                <xs:element name="staffID" type="OIDType"/>
                <!-- additional nesting -->
                <xs:element name="managedBy">
                    <xs:key name="manager">
                        <xs:selector/>
                        <xs:field xpath="staffID"/>
                </xs:key>
            </xs:element>
        </xs:sequence>
    </xs:complexType>
    <!-- additional nesting -->
    <!-- additional nesting -->
    <xs:complexType name="OIDType">
        <xs:sequence>
            <xs:element name="ID">
                <xs:unique name="OID">
                    <xs:selector xpath="OIDType"/>
                    <xs:field xpath="ID"/>
                </xs:unique>
            </xs:element>
        </xs:sequence>
    </xs:complexType>
   <!-- additional nesting -->
```

Code Listing 7.2: An example transformation of uniqueness constraint

Example 7.26: The ordered/unordered compositions (e.g. in Fig. 7.15, `Lot-Master` & `Goods-Items`), shown using the stereotype (`<<1>>, <<2>>,...`), are mapped using the `<xs:sequence>` construct. The code fragment in Code Listing 7.3 illustrates this.

```
<!-- additional nesting -->
<xs:complexType name="Goods-TransactionType">
    <xs:complexContent>
        <xs:extension base="Goods-TransactionType">
            <!-- additional nesting -->
            <xs:sequence>
                <xs:element name="Lots-Master" type="Lots-MasterType"
                                            minOccurs="1" maxOccurs="unbounded"/>
                <xs:element name="Goods-Items" type="Goods-ItemsType"
                                            minOccurs="1" maxOccurs="unbounded"/>
            </xs:sequence>
            <!-- additional nesting -->
        </xs:extension>
    </xs:complexContent>
</xs:complexType>

<!-- additional nesting -->
<xs:complexType name="Lots-MasterType">
    <!-- additional nesting -->
</xs:complexType>

<!-- additional nesting -->
<xs:complexType name="Goods-ItemsType">
    <!-- additional nesting -->
</xs:complexType>
<!-- additional nesting -->
```

Code Listing 7.3: An example transformation of ordered constraint

Example 7.27: As shown in Fig. 7.15, the exclusive disjunction constraint between `Internal-Lot-Movement` and `External-Lot-Movement` can be mapped XML Schema using the `<xs:choice>` schema construct, as shown below in Code Listing 7.4.

Example 7.28: In the e-Sol, the `Goods-Sub-Type` has an attribute which is `Categories` that may contain 1 to 5 `Sub-Categories`. Code Listing 7.5 illustrates this.

Example 7.29: In the e-Sol, the `Internal-Lot-Movement` date may be specified using a typical date format or using the week number of the year. The code fragment in Code Listing 7.6 illustrates this.

```
<!-- additional nesting -->
<xs:element name="Lot-Movement">
<xs:complexType>
<xs:complexContent>
                        <xs:extension base="LotMovementType">
                        <!-- additional nesting -->
                          <xs:choice>
                        <xs:element name = "Internal-Lot-Movement" type="InternalLotMovementType"/>
            <!-- additional nesting -->
            <xs:element name= "External-Lot-Movement" type="ExternalLotMovementType"/>
                        <!-- additional nesting -->
                          </xs:choice>
                        <!-- additional nesting -->
                        </xs:extension>
            </xs:complexContent>
</xs:complexType>
</xs:element>
<!-- additional nesting -->
```

Code Listing 7.4: An example transformation of exclusive disjunction constraint

```
<!-- additional nesting -->
    <xs:simpleType name="Sub-Categories">
        <xs:list itemType="xs:string"/>
    </xs:simpleType>
    <!-- additional nesting -->

    <!-- additional nesting -->
    <xs:simpleType name="Categories">
        <xs:restriction base="Sub-Categories">
            <xs:length value="5"/>
        </xs:restriction>
    </xs:simpleType>
<!-- additional nesting -->
```

Code Listing 7.5: An example transformation of list type and properties

```
<!-- additional nesting -->
    <xs:simpleType name="Lot-Movement-DateType">
        <xs:union memberTypes="xs:date xs:integer"/>
    </xs:simpleType>
<!-- additional nesting -->
```

Code Listing 7.6: An example transformation of union type

Table 7.1: Summary of conceptual views (UML/OCL) to logical views (XML Schema)

XML Conceptual Views (UML/OCL)	**XML Schema**
Conceptual view and view class hierarchy (View-of-view)	Complex type declaration construction (xs:complexType) and complex type/element hierarchies
Structural Relationships (IS-A, composition, association)	<xsd:sequence>, <xsd:choice> & <xsd:all> with <xsd:extension> / <xsd:restriction> and ID, IDREF/IDREFS or KEY and KEYREF constructs
View attributes	XML Schema Simple types (integer, float, string, Boolean, date, time etc.)
View class constraints Ordering, homogenous composition	Built-in XML Schema constraints.
View attribute constraints (e.g. Object Identifier (OID)	XML Schema constructs using "facet" and XML Schema ID, IDREF/IDREFS or KEY and KEYREF constructs
Attribute grouping (constructional contents such as set, bag)	XML Schema list (NMTOKENS, IDREFS, & ENTITIES), or new list types using <xsd:list> construct and union by using <xsd:union> construct.
Structural Relationships (IS-A, composition, association)	<xsd:sequence>, <xsd:choice> & <xsd:all> with <xsd:extension> / <xsd:restriction> and ID, IDREF/IDREFS or KEY and KEYREF constructs
Domain constraints	"facets" mechanism to create new types, apply restriction etc.
Uniqueness Constraint	XML Schema <xsd:unique> construct
Cardinality Constraint	XML Schema maxOccurs/minOccurs construct
Referential Constraint	XML Schema ID, IDREF/IDREFS or KEY and KEYREF constructs.

7.6 Conclusion

In this chapter we presented another modelling and transformation methodology of conceptual views in the LVM using UML/OCL. This chapter is supplementary to Chapters 5 and 6, where conceptual view semantics and properties are clearly defined and described. Here, our main focus was on modelling UML modelling notations for conceptual views rather than conceptual view concepts or UML notations, as the first was discussed in Chapters 4, 5 and 6 and the base set of UML notations are well understood and practised. We also provided a detailed discussion and illustrated examples from the e-Sol case study to describe additional conceptual view properties (those that cannot be represented visually in UML or for UML notational semantics which are poorly defined or semantically poor to specify conceptual views) using OCL expressions.

In the following chapters, specifically in Chapters 8 and 10, the modelling and transformation concepts presented in this chapter are utilized to develop architectural frameworks for two real-world application scenarios, namely, in modelling and transformation of dimensional data in an XML document warehouse framework (Chapter 8) and website design in a view-driven, web engineering framework (Chapter 10).

References

Balsters, H 2003, 'Modelling Database Views with Derived Classes in the UML/OCL-framework', *The Unified Modeling Language: Modeling Languages and Applications (UML '03)*, vol. 2863, Springer, USA, pp. 295-309.

Binstock, C & al., e 2002, *The XML Schema Complete Reference*, Addison-Wesley, Boston.

Booch, G 1993, *Object-oriented analysis and design with applications*, 2nd edn, The Benjamin/Cummings series in object-oriented software engineering., Benjamin/Cummings Pub. Co.; Addison-Wesley, Redwood City, Calif. Reading, Mass.

Coad, P & Yourdon, E 1991, *Object-oriented analysis*, 2nd edn, Yourdon Press computing series., Prentice Hall, London.

Dillon, TS & Tan, PL 1993, *Object-Oriented Conceptual Modeling*, Prentice Hall, Australia.

Feng, L, Chang, E & Dillon, TS 2002, 'A Semantic Network-based Design Methodology for XML Documents', *ACM Transactions on Information Systems (TOIS)*, vol. 20, No 4, no. 4, pp. 390 - 421

Feng, L, Chang, E & Dillon, TS 2003, 'Schemata Transformation of Object-Oriented Conceptual Models to XML', *International Journal of Computer Systems Science & Engineering*, vol. 18, No. 1, no. 1, pp. 45-60

OMG-OCL 2003, *UML 2.0 OCL Final Adopted specification (http://www.omg.org/cgi-bin/doc?ptc/2003-10-14)*, OMG, 2005.

OMG-UML™ 2003, *UML 2.0 Final Adopted Specification (http://www.uml.org/#UML2.0)*, OMG, 2005.

Rahayu, JW 2000, 'Object-Relational Transformation Methodology', Doctor of Philosophy (Ph.D) thesis, La Trobe University, Melbourne, Australia.

Sergio Lujan-Mora, Juan Trujillo & Song, I-Y 2002a, 'Extending the UML for Multidimensional Modeling', *Fifth International Conference on the Unified Modeling Language and its applications (UML '02)*, Springer-Verlag London, UK, Dresden, Germany, pp. 290-304.

Sergio Lujan-Mora, Juan Trujillo & Song, I-Y 2002b, 'Multidimensional Modeling with UML Package Diagrams', *Proceedings of the 21st International Conference on Conceptual Modeling (ER '02)*, Springer-Verlag London, UK, pp. 199-213.

W3C-XSD 2001, *XML Schema (http://www.w3.org/XML/Schema)*, W3C, 2004.

Warmer, JB & Kleppe, AG 2003, *The object constraint language : getting your models ready for MDA*, 2nd edn, Addison-Wesley, Boston, MA.

Xiaou, R, Dillon, TS, Chang, E & Feng, L 2001a, 'Mapping Object Relationships into XML Schema', *Proceedings of OOPSLA Workshop on Objects, XML and Databases.*

Xiaou, R, Dillon, TS, Chang, E & Feng, L 2001b, 'Modeling and Transformation of Object-Oriented Conceptual Models into XML Schema', *12th International Conference on*

Database and Expert Systems Applications (DEXA '01) 2001, vol. 2113, ed. JL Heinrich C. Mayr, Gerald Quirchmayr, Pavel Vogel, Springer.

Chapter 8

Case Study & Applications I: View-Driven XML Document Warehouse Model

8.1 Introduction

In the previous chapters, a conceptual framework for modelling, designing and transforming views in the Layered View Model (LVM) was presented. In this chapter (and the following two chapters) we present a real-world case study and applications scenarios where the views in the LVM are applied. Here, in this chapter, a conceptual framework for designing an XML Document Warehouse (XDW) is presented. The development of the XDW model and architecture is the subject of a separate PhD thesis (Vicky Nassis 2006). Here we intend primarily to demonstrate the use of views to support the XDW framework.

The XDW is an *architectural framework* that is based on the LVM concept and enables us to model dimensional XML data/documents at varying levels of abstraction. In contrast to the traditional data warehouse conceptual model, the *view-driven* XDW model, warehouse FACT and associated dimensions are considered at three levels of abstractions, namely, the conceptual, logical and document levels. Also, the design of the dimensional model is focused on capturing and modelling user requirements (Vicky Nassis 2006), in contrast to modelling available operational data semantics. Due to the strong motivation for this work and the wide-spread interest it received in various research and academic forums, an extension of this work is referenced in (Vicky Nassis 2006), in the form of *Requirements Based and*

Conceptual Approach for XML Document Warehouse Analysis and Design XDW design.

It should be noted that, though we refer to an XML data warehouse as an XML Document Warehouse (XDW), the concepts presented here are common to both *data-centric* and *document-centric* XML documents.

This chapter is organized as follows. In section 8.2, the motivation for this research (i.e. XDW) is presented followed by a general introduction to the notion of data warehouse and the related concepts are given in section 8.3. Section 8.4 discusses some of the background and related work in this area. In section 8.5, we provide an extended discussion of the illustrated case study example (e-Sol) that was originally described in Chapter 4 and used in the chapter, in the context of XDW. Sections 8.6 – 8.9 provide detailed a discussion of the XDW model, concepts, modelling issues and definitions. Section 8.10 concludes this chapter.

8.2 Motivation

In recent years, data warehouse models have focused on incorporating conceptual semantics and user requirements as part of the model specification. This kind of work is still in the early stages; only a few such works consider conceptual semantics (in contrast to operational data oriented classical data warehouse designs such as the Star Schema (Kimball & Ross 2002) model) and are focused on both conceptual and user requirements as part of the DW design process.

In this chapter, we present our approach to this problem: a *view-driven* conceptual framework for developing dimensional models for XML documents. Our work is radically different from existing works such as (Gopalkrishnan, Li & Karlapalem 1999; Jeong & Hsu 2001; Lucie-Xyleme 2001; Luján-Mora, Trujillo & Vassiliadis 2004; Luján-Mora, Vassiliadis & Trujillo 2004; Medina, Luján-Mora & Trujillo 2002; Mohammed 2001; Mohania, Karlapalem & Kambayashi 1999). In this research, we look at views as the foundation for developing dimensional architectural constructs rather than representing aggregate and/or dimensional queries (and query wrappers). Also, the design of the dimensional model is focused on capturing and modelling user requirements (Vicky Nassis 2006), in contrast to developing dimensional models using available operational data and the associated semantics (Kimball & Caserta 2004; Luján-Mora & Trujillo 2004). In summary, the main motivation for XDW research includes:

(i) *User requirements*: separation of operational data semantics and data warehouse user requirements in the case of XML data and document

(ii) *Top-down approach*: separation of implementation concerns (data format, structure, etc.) from (XML) data warehouse conceptual models

(iii) *Expressiveness*: formulation of dimensional semantics that are capable of expressively modelling XML data (both data and document centric) semantics

(iv) *XML*: providing dimensional semantics that can be expressed and described using XML (and XML Schema) itself

(v) *Views in the LVM*: investigating application of the LVM for dimensional modelling in achieving (ii) – (iv) above.

Another motivation is the design of DW using conceptual semantics such as in OMG's Model-Driven Architecture (MDA) initiative (OMG-MDA 2003). Since the introduction of the MDA initiative, platform independent models play a vital role in system development and data engineering. Under the MDA initiative, first the model of a system is specified via an abstract notation independent of the technical or deployment specifications (i.e. Platform Independent Model or PIM), and then the PIM is mapped or transformed into a deployment model (i.e. Platform Specific Model or PSM) by adding platform or deployment specific information into the PIM. To support MDA initiatives in ECM (i.e. data engineering, data semantics, constraints etc.), model requirements have to be specified precisely at a higher level of abstraction. This presents an opportunity to investigate conceptual views as a means of providing data abstraction and semantics in PIMs for data intensive MDA solutions.

8.3 Data Warehouses

Data Warehousing (DW) has been an approach adopted for handling large volumes of historical data for detailed analysis and management support. Transactional data in different databases is cleaned, aligned and combined to produce good data warehouses. At the most basic level, data warehousing has been an approach adopted for management of large volumes of historical data for detailed analysis to provide crucial business intelligence (BI) for organisations in:

(i) Decision Support Systems (DSS) (Elmasri & Navathe 2004; Gray & Watson 1998),

(ii) Management Information Systems (MIS) (Elmasri & Navathe 2004) and

(iii) Executive Information Systems (Gray & Watson 1998).

A data warehouse integrates large amounts of enterprise data from multiple and independent data sources consisting of operational databases into a common repository (Feng & Dillon 2003) for querying and analysis (using BI tools). In addition, data warehouses are designed for online analytical processing (OLAP) (Elmasri & Navathe 2004; Feng & Dillon 2003; Kimball & Ross 2002; Trujillo, Luján-Mora & Song 2003), where the queries aggregate large volumes of data in order to detect trends and anomalies. To reduce the cost of executing aggregate queries in such an environment, warehousing systems usually pre-compute frequently used aggregates and store each materialized aggregate view (Feng & Dillon 2003; Gopalkrishnan, Li & Karlapalem 1999; Theodoratos & Sellis 1999) in a multidimensional data cube (Feng & Dillon 2003; Gopalkrishnan, Li & Karlapalem 1999; Gupta, Mumick & (eds) 1999; Trujillo, Luján-Mora & Song 2003). These data cubes group the base data along various dimensions, corresponding to different sets of operational attributes, and compute different aggregate functions (e.g. sum, avg, min, max) on measures.

In traditional data warehouse terminology, the dimensional model is represented using FACTs and dimensions. A FACT is a business performance measurement usually numeric in nature and a dimension refers to an independent entity that serves as an entry point and/or mechanism to extract meaningful measurements form the associated FACT (Elmasri & Navathe 2004; Kimball & Caserta 2004; Kimball & Ross 2002). Also, depending on the dimensional model (i.e. OO (Giovinazzo 2000), Star Schema, O-R star (Mohammed 2001)) further terminologies (e.g. FACT dimensions, shared dimensions, etc.) are defined to elaborate some of additional features of those models. In the relational model, the popular dimensional model is the Kimbal et al. Star Schema model (Kimball & Ross 2002), where the FACT and the dimensions are represented using tables (and materialized views), referred to as FACT and dimension tables. A FACT table is a non-normalized table with the numeric performance measurements characterized by a group of (foreign) keys drawn from the dimensional tables that form a composite (foreign key).

Since its introduction in 1996, eXtensible Markup Language (XML) (W3C-XML 2004) has become the *defacto* standard for storing and manipulating self-describing information (meta-data), which creates vocabularies in assisting information exchange between heterogenous data sources over the web (Pokorn'y 2002). Due to this, there is considerable work to be achieved in order to allow electronic document handling, electronic storage, retrieval and exchange. It is envisaged that XML will also be used for logically encoding documents for many domains. Hence, it is likely that a large number of XML documents will populate the would-be repository and several disparate transactional databases. Conversely, Enterprise Content Management (ECM) is the integration and utilization of one or more technologies, tools, and methods to capture, manage, store and deliver content across an enterprise (ECM-AIIM 2005), where XML is gaining momentum as the data representation and integration language. One of the data intensive issues in ECM is the data warehousing concept that has gained in importance in recent years (Elmasri & Navathe 2004).

The concern of managing large amounts of XML document data raises the need to explore the data warehouse approach through the use of XML document marts and XML document warehouses. Since the introduction of dimensional modelling which, evolves around facts and dimensions, several design techniques have been proposed to capture multidimensional data (MD) at the conceptual level. Ralph Kimball's Star Schema (Kimball & Ross 2002) proved most popular, from which well-known conceptual models SnowFlake and StarFlake were derived. More recent comprehensive data warehouse design models are built using Object-Oriented concepts (structural relationships, Object cubes or data cubes) on the foundation of Star Schema. In (Abelló, Samos & Saltor 2001; Lujan-Mora, Trujillo & Song 2002a, 2002b; Luján-Mora, Trujillo & Vassiliadis 2004; Trujillo et al. 2001) two different OO modelling approaches are demonstrated where a data cube is transformed into an OO model integrating class hierarchies. The Object-Relational Star schema (O-R Star) model (Mohammed 2001; Rahayu et al. 2001) aims to envisage data models and their object features, focusing on hierarchical dimension presentation, differentiation and their various sorts of embedded hierarchies.

These models, both object and relational, have a number of drawbacks, namely:

(i) data-oriented without sufficient emphasis or capturing user requirements;

(ii) extensions of semantically poor relational models (star, snowflake models);

(iii) original conceptual semantics are lost before building data warehouses as the operational data source is relational;

(iv) further loss of semantics resulting from oversimplified dimensional modelling;

(v) time consuming if additional data semantics are required to satisfy evolving user requirements; and

(vi) complex query design and processing is needed, therefore maintenance is troublesome.

In applying these approaches to the design of XML document warehouses, it is important to consider XML's non-scalar, set-based and semi-structured nature. Traditional design models lack the ability to utilise or represent XML design level constructs in a well-defined abstract and implementation-independent form.

8.4 Background

Since the introduction of dimensional modelling, several design techniques have been proposed to capture multidimensional data (MD) at the conceptual level. Conceptual data models for such MD, which revolves around FACTs, dimensions and hierarchies, have been extensively discussed in research and industrial literature (Feng & Dillon 2003; Trujillo, Luján-Mora & Song 2003). These discussions normally include support for data warehouses and OLAP data (ROLAP, MOLAP), where MD is the feasible data model for such applications.

The early work on MD and data warehousing concepts dates back to works done by W.H. Inmon et al. (Gray & Watson 1998; Humphries, Hawkins & Dy 1999; Inmon, Imhoff & Battas 1996; Jacek Blazewicz et al. 2003). Later work by Ralph Kimball's popular Star Schema (Elmasri & Navathe 2004; Kimball & Ross 2002) provided the base for other well-known conceptual models such as SnowFlake and StarFlake to be derived. More recent comprehensive data warehouse design models are built using Object-Oriented concepts on the foundations of the Star Schema. In (Lujan-Mora, Trujillo & Song 2002a, 2002b; Luján-Mora, Trujillo & Vassiliadis 2004; Luján-Mora, Vassiliadis & Trujillo 2004; Trujillo, Luján-Mora & Song 2003; Trujillo et al. 2001), and (Abelló, Samos & Saltor 2001), two different OO modelling approaches are demonstrated where a data cube is transformed into an OO model integrating class hierarchies. The Object-Relational Star schema (O-R Star) model (Rahayu et al. 2001) aims to utilise Object-Relational (O-R) data model and its features for warehouse MD data and hierarchies.

For XML data, one of the early XML data warehouse implementations for web data includes the Xyleme Project (Lucie-Xyleme 2001). The Xyleme project (Xyleme 2001) was successful and it was made into a commercial product in 2002. It has well defined implementation architecture and proven techniques (such as materialised views) to collect and archive web XML documents into an XML warehouse for further analysis. Another approach by Fankhauser et al. (Fankhauser & Klement 2003) explores some of the changes and challenges of a document-centric XML warehouse. Other works that use XML in a data warehouse context include (Golfarelli, Rizzi & Vrdoljak 2001; Medina, Luján-Mora & Trujillo 2002).

In DW design, views are mainly used to provide aggregate data and queries, performance (as materialized views), meta-data and OLAP queries (Debevoise 1999; Elmasri & Navathe 2000; Gray & Watson 1998; Gupta, Mumick & (eds) 1999; Humphries, Hawkins & Dy 1999; Johnson 2002; Kimball & Ross 2002; Ponniah 2001; Trujillo & Luján-Mora 2003; Trujillo, Luján-Mora & Song 2003; Trujillo et al. 2001). Little work has been done in the direction of using views for providing DW architectural constructs and frameworks (Gopalkrishnan, Li & Karlapalem 1999; Mohania, Karlapalem & Kambayashi 1999; Theodoratos & Sellis 1999).

Our research is different from other work such as (Cluet, Veltri & Vodislav 2001; Kimball & Ross 2002; Lucie-Xyleme 2001; Mohania, Karlapalem & Kambayashi 1999; Ponniah 2001). Here, the LVM views are used to model and design dimensional data instead of using views for the purpose of providing data granularity, dimensional refinements and/or for performance (e.g. materialized views). The XDW is designed for XML data and documents by incorporating XML specific data semantics.

It is different from approaches such as OMG's CWM approaches (OMG-CWM 2001) where metadata model specifications based on MOF (OMG-MOF™ 2003) were proposed for developing a (mostly relational) data warehouse conceptual model. Conversely, the works such (Lujan-Mora, Trujillo & Song 2002a, 2002b; Luján-Mora, Trujillo & Vassiliadis 2004; Luján-Mora, Vassiliadis & Trujillo 2004; Trujillo, Luján-Mora & Song 2003; Trujillo et al. 2001), where conceptual models of a (mostly relational) data warehouse are developed using Object-Oriented (OO) techniques and languages were proposed. Though they are similar to our work from a conceptual modelling point of view, the works do not include:

- an architectural framework to develop a common framework for different data domains;
- explicit data warehouse requirement specification and notational representation of such user requirements;
- constructs to model semi-structured (e.g. XML) data model specific semantics such as ordering. But in the work, some new directions have

been proposed to support OO concepts in the traditional FACT driven data warehouse models.

As stated in Chapter 2, our research is also different from the work in (Cluet, Veltri & Vodislav 2001; Lucie-Xyleme 2001; Xyleme 2001), where an implementation (web) warehouse architecture is proposed for archiving web XML documents as a web document warehouse. In the Xyleme project, the use of the declarative XML views notion is analogous to the usage of relational views (and materialized views) in the (relational) Star Schema (or variant) model.

8.5 Example Case Study

As an illustrative case study for this chapter, we intent to gradually build our e-Sol case study described in Chapter 4 as a XDW conceptual and logical model for the purposes of archiving and analysing e-Sol data for the purpose of business planning and intelligence. We refer to the e-Sol XDW model as e-Sol-W. Given below is the extended description and requirements of the e-Sol-W for the purpose of building a data warehouse and data marts.

For e-Sol to support DSS, EIS and MIS, it is essential to provide a data model to support dimensional data in the context of a data warehouse. Due to e-Sol's dynamic and heterogeneous nature (both system and data, as described in Chapter 4), the data warehouse model should support rapidly evolving new data formats (from relational, XML to propriety data scripts), at a higher-level of abstraction. From a local stakeholder/partners' perspective, the XDW model solves some of the problems faced by e-Sol. But from a global perspective, where multiple stakeholders/partner systems are involved (i.e. collaborative partners, global customers, etc.) and there is a need to support e-Sol's global information demand, the role and scope of XDW has to evolve and a new global warehouse model is inevitable and an unfortunate reality.

To illustrate our concepts, we highlight a few, simplified XDW user requirements. We consider a simplified XDW model for archiving and analysing warehouse bookings, income and capacity for the warehouses in the e-Sol. Some of these requirements include:

(i) Warehouse booking: Warehouse booking records by (a) customers, (b) companies and (c) collaborative partners, grouped by (i) year, (ii) month and (iii) by warehouse location or by region (e.g. Asia-Pacific, China etc.). This information may help to rate customers and to plan warehouse capacity around the world, for a given time of the year.

(ii) Warehouse Capacity: Sort warehouse usage (i.e. slack space and near-full capacity measure) by year, month and quarterly and individual Q1, Q2, Q3 and Q4 capacity measure for (a) warehouse by region, and (b) warehouse by country.

(iii) Warehouse Revenue: List warehouse revenue by (a) year, (b) month (c) quarterly and (d) individual Q1, Q2, Q3 and Q4 for (i) individual warehouses, and (b) warehouses by region.

8.6 The XML Document Warehouse Model (XDW)

The XDW model proposed in this research together with Nassis et al. (Vicky Nassis 2006), is composed of four design levels, namely:

(i) XDW requirements level

(ii) XDW conceptual level

(iii) XDW logical (or schema) level

(iv) XDW document (or instance) level

Here, except for the requirements level, the other three levels are analogous to the layers of abstraction in the LVM. The XDW requirements level, in addition to the layers of abstraction, enforces user requirements, capturing and representing these in the form of an XDW Warehouse Requirements (WR) model and User Requirement (UR) model (Vicky Nassis 2006), which supplements the XDW conceptual model. A context diagram of this model is given in Fig. 8.1.

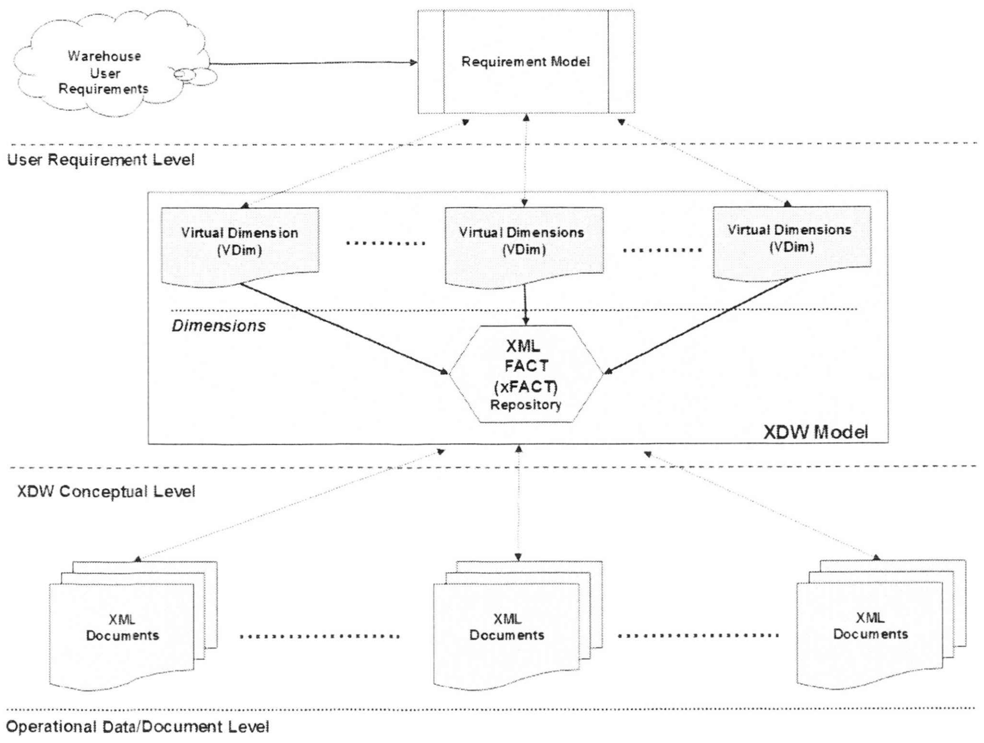


Figure 8.1: XDW Model (context diagram)

The first design level, the user requirement level, has two components:

(a) warehouse requirements, and

(b) user requirements model

The second design level is the development of the XDW conceptual model that has two main components, namely:

(a) the XML FACT Repository (*x*FACT)

(b) the Virtual Dimensions (VDim)

The third level is the development of the logical model of the XDW, namely the schemata transformation of the *x*FACT and the associated VDims to (XML) schemas. The fourth level is the transformation of VDim construct (i.e. conceptual operators) to document level query expressions using one or more native or embedded query languages (e.g. XQuery).

In this chapter, we focus only on the conceptual and logical levels of the XDW, where the views of the LVM are used to specify and define the dimensional (XML) data model. Thus, the conceptual and logical model design and transformation of the XDW is analogous to the conceptual and logical levels (and the transformations) of the LVM. We do not discuss the first level which deals with the requirements because that is outside the scope of this thesis and is part of another PhD thesis in (Vicky Nassis 2006).

In addition to being designed and developed using the LVM, the uniqueness of the XDW model is also in its approach to capturing and specifying data warehouse requirements (Vicky Nassis 2006; Vicky Nassis et al. 2006 - *to appear;* Vicky Nassis et al. 2006; Vicky Nassis et al. 2005a). This is because, traditionally, a data warehouse model is heavily constrained by the available operational data and its structure, as warehouse modellers and designers design a data warehouse using bottom-up approaches (or reverse engineer warehouse requirements), working from operational data to the warehouse conceptual model.

Thus, it should be noted here that, in comparison with traditional data warehouse requirements, as unique to XDW design, we first develop the UR model using specialized notations (developed in (Vicky Nassis 2006)) that are independent of the operational data and/or data structures. Also, the UR model is developed first before constructing the conceptual model of the XDW, but it is *iteratively* validated against the available operational data (conceptual) model and/or structures. By adopting this approach, we intend to model and represent user requirements that are *valid* yet independent of the operational data.

Also, in addition to adopting user requirement driven XDW design, the XDW model outlined below, to our knowledge, is unique in its kind as it is utilizes XML itself (together with XML Schema) to provide;

(i) structural constructs,

(ii) metadata,

(iii) validity, and

(iv) expressiveness (via refined granularity and class decompositions).

8.6.1 XDW Conceptual Level

As stated earlier, the XDW conceptual model is composed of: (a) an XML FACT repository (*x*FACT); and (b) a collection of associated, logically grouped ***conceptual views***. That stratifies the one or more user requirements given in the UR model. The *x*FACT is a snapshot of the underlying transactional system(s) for a given *context*.

As defined earlier in Chapter 5, a *context* is more than a measure or an item that is of interest for the organization as a whole. In classical data warehouse models, a context is normally modelled as an ID packed FACT and associated data perspectives as dimensions. Usually, due to constraints of the relational model, a FACT will be collapsed to a single table, with IDs of its dimension(s), thus emulating (with combination of one or more dimension(s)) a data cube (or dimensional data). A complex set of queries are needed to extract information from the FACT-Dimension model. But, in regards to XML, a context is more than a flattened FACT (or simply referred to as *meaningless* FACT) with embedded semantics and constraints. It will also have embedded relationships such as those featured in OO models and semi-structured data such ordered composition, exclusive disjunction etc. in addition to non-relational constructs such as set, list, and bag. Therefore, we argue that, a FACT structure similar to a (relational) FACT table without the required semantics does not provide semantic constructs that are needed to accommodate an XML *context*.

The role of conceptual views is to provide perspectives to the document hierarchy stored in the *x*FACT repository. Since conceptual views can be grouped into logical groups, each group is very similar to that of a ***subject area*** (or class categories) (Dillon & Tan 1993) in OO conceptual modelling techniques. Each subject-area in the XDW model is referred to as a cluster of *Virtual Dimensions* (VDim) in accordance with dimensional models. VDim is called *virtual*; that is, since it is modelled using XML *conceptual views* (which are *imaginary* XML documents) in the LVM and behaves as a dimension for the given *x*FACT.

8.6.1.1 The XML FACT Repository

An XML FACT repository (*x*FACT) is a hierarchical decomposing of the operational data model that satisfies a given warehouse requirement. For example, collection and sort of yearly conferences for the purpose of archiving and searching is an example of such a warehouse requirement that may require a detailed design of *x*FACT. It is a snapshot of the underlying transactional system/(s) for a given *context*. A context is a meaningful collection of classes and relationships, which is designed to provide various perspectives (with the help of one or more *conceptual views*) for business intelligence, as defined in Chapters 4 and 5, in the case of the LVM.

In building the *x*FACT, we vary from modelling the traditional data warehouse flat FACT tables by adding semantics. That is to say, the *x*FACT is more semantically descriptive due to its interconnected relationships and decomposed hierarchical structures (Fig. 8.3). The relationships considered in *x*FACT are not restricted to association (1:1, 1:m, n:m) relationships, but also homogenous composition (with shared aggregation with cardinality 1:n and n:m) specialization/generalization and ordering relationships that are analogous to the discussion provided for the LVM.

At the logical level, XML schema definition language is used to describe the *x*FACT semantics (classes, relationships, ordering and constraints). The logical model of the *x*FACT is achieved by applying schemata transformations of the *x*FACT conceptual model by applying transformation rules, as discussed in Chapters 6 -7, in the context of LVM.

Though *x*FACT may be modelled and implemented as a standalone (XML) repository by aggregating and decomposing operational data (and associated data semantics) during the warehouse Extract-Transform-Load (ETL) phase, and can be intuitively done using (materialized and/or) views based on the LVM.

8.6.1.2 View-Driven *x*FACT

Intuitively, it can be seen that one can model and design an *x*FACT using conceptual views themselves. That is, an *x*FACT may be considered as a collection of hierarchically organized conceptual views. A *conceptual view* driven *x*FACT model is referred to in this research as a Global XML FACT Repository (or $\mathcal{G}$xFACT in short). That is, it is a global (collection) of conceptual views. Section 8.9 presents a detailed discussion of $\mathcal{G}$xFACT.

8.6.1.3 Virtual Dimensions

A user requirement, which is captured and specified in the XDW requirement model (namely UR model), is transformed into one or more conceptual views in the LVM, which are referred to as *Virtual Dimension/s*, (VDim) in association with the *x*FACT. These are typically *views* involving aggregation or *perspectives* of the underlying *x*FACT, which serves as the pre-defined context.

A valid user requirement is such that, it can be satisfied by one or more XML conceptual views for a given context (i.e. *x*FACT). But in the case where for a given user requirement there is no transactional document or data fragment to satisfy it, further enhancements are necessary to make the requirement feasible to model with a certain *x*FACT. Therefore, modelling and specifying VDim is an iterative process, where user requirements are validated against the *x*FACT in conjunction with the operational data and data semantics. Thus, VDim is an additional elaboration, extraction and/or specification of the required information from the *x*FACT (thus, from the aggregated operational data).

VDim can be materialized (for data refinement or for the purpose of performance issues such as relational views in classical Star model) or aggregated further by defining additional conceptual views to refine and/or satisfy further user requirements.

8.7 XDW Model Semantics

In this section, we present some of the formal semantics associated with the XDW model (without the UR model). Since the proposed model is driven by LVM, some of the concepts and definitions are extensions and/or an elaboration of the concepts and definitions presented in Chapter 5.

The XDW conceptual model consists of an *x*FACT repository and multiple hierarchical dimensions. Thus, at first glance, the XDW is analogous to the Star/Snowflake schema (Gopalkrishnan, Li & Karlapalem 1999) of the relational model, or the Operational Data Store (ODS) (Inmon, Imhoff & Battas 1996) model, except that both the *x*FACT and the VDims are modelled using views in the LVM. Also, since *x*FACT is more complex than a relational FACT table, it can be considered as one context (e.g. sales) that is of interest to the organization, with multiple sub-contexts (such as regional-sales, sales-by-city, sales-by-store etc.). Therefore it can be shown that there exists a many-to-many (m:n) relationship between one *x*FACT and a VDim (or a VDim hierarchy).

8.7.1 xFACT and View-Driven VDim

Let XML FACT repository (*x*FACT) be denoted as x_{FACT} and a virtual dimension (VDim) denoted as V^{Dim} in the XDW model. In Chapter 5 we defined a *conceptual view* using a *context*. Since each VDim is a conceptual view, from Chapter 5 definition 5.2, a virtual dimension V^{Dim} can be defined as :

Definition 8.1. A virtual dimension is a conceptual view V^{Dim}, such that V^{Dim} is a 4-ary tuple of $V_{name}^{Dim}, V_{obj}^{Dim}, V_{rel}^{Dim}$ and $V_{constraint}^{Dim}$, where V_{name}^{Dim} is the name of the virtual dimension V^{Dim}, V_{obj}^{Dim} is a set of objects in V^{Dim}, V_{rel}^{Dim} is a set of object relationships in V^{Dim}, and $V_{constraint}^{Dim}$ is a set of constraints associated with V_{obj}^{Dim} and V_{rel}^{Dim} in V^{Dim}.

$$V^{Dim} = (V_{name}^{Dim}, V_{obj}^{Dim}, V_{rel}^{Dim}, V_{constraint}^{Dim}) \qquad \textbf{(8.1)}$$

If one considers an xFACT to be a context, it can be shown as in definition 8.2, below using the definition of context presented in Chapter 5 (definition 5.1) as:

Definition 8.2. An XML FACT (*x*FACT) repository xF is defined such that, xF consists of a *x*FACT name xF_{name}, a set of objects xF_{obj},, a set of object relationships xF_{rel}, and a set of constraints associated with its objects and relationships $xF_{constraint}$.

$$xF = (xF_{name}, xF_{obj}, xF_{rel}, xF_{constraint}) \qquad \textbf{(8.2)}$$

Similar to the definition of a valid conceptual view, here we can define a valid virtual dimension as:

Definition 8.3. A virtual dimension V^{Dim} called a *valid* virtual dimension for a given XML FACT (*x*FACT) repository xF, if and only if for any object $\forall obj \in V_{obj}^{Dim}$, there exist objects $\exists obj_1, obj_2, ..., obj_n \in xF_{obj}$, such that $obj = \lambda_1 .... \lambda_m (obj_1, ..., obj_n)$, where $\lambda_1 .... \lambda_m \in \hat{\lambda}$ and $\hat{\lambda}$ be a set of conceptual operators. That is, obj is a newly derived object from existing objects $obj_1, obj_2, ..., obj_n$ in xF via a series of conceptual operators $\lambda_1 .... \lambda_m$.

From definition 8.3, it is intuitively deductible that, an xF for a given V^{Dim} is actually the context for the in V^{Dim} question, as defined in Chapter 5.

8.7.2 XDW Relationships

Typically, in an XDW model, for one *x*FACT, there exists one or more VDims. Let the total number of V^{Dim} be ***n***. Let ${}_xR_d$ denote the relationship between the *x*FACT and a VDim and ${}_dR_d$ between two dimensions. The relationship between *x*FACT xF and VDim V_k^{Dim}, may be denoted as:

$${}_xR_d^{\;k} = (xF, V_k^{Dim}) \qquad \textbf{(8.3)}$$

where $0 < k \le n$.

Also, the hierarchical relationship (dimensional hierarchy) between two VDims, V_k^{Dim} and V_{k+1}^{Dim} can be shown as:

$${}_dR_d^{\;k} = (V_k^{Dim}, V_{k+1}^{Dim}) \qquad \textbf{(8.4)}$$

Example 8.1: For example, as shown in Fig. 8.2, in the e-Sol-W case study, a VDim hierarchy, where there exist inheritance relationships between the VDim `Quarterly_WarehouseCapacity` and the individual Q1, Q2, Q3 and Q4 warehouse capacities.

Both ${}_xR_d$ and ${}_dR_d$ may fall into one or more of the dimension specific relationships type (and constraints) as:

- aggregate dimension (minimum , maximum, count, average),
- time-variant dimension

- subject-variant dimension
- aggregate-descriptive dimension

These types of relationships may correspond to one or more of the dimensional (conceptual) operators or queries, as described in (Vicky Nassis 2006; Vicky Nassis et al. 2005b). They may be grouped into:

- selection
- sort
- implicit join
- explicit join
- associated

Analogous to the LVM, at the logical level, these dimensional (conceptual) operators or queries are transformed into W3C use case query context algorithms and later at the document/instance level to language specific query expression, such as XQuery and/or SQL.

8.8 XDW Modelling and Transformation

In our research, we use UML as the OO modelling language to represent XDW conceptual level artefacts. Hence, we use of UML just as a modelling notation as it is easily understood and standardised. As we have stated in our work in LVM, other OO languages may also be used instead of UML.

To model and represent XDW concepts, we utilize UML stereotypes. We introduced a new UML stereotype called `<<VDim>>` to model the virtual dimensions at the XDW conceptual level. Analogous to conceptual views in the LVM, this stereotype is similar to a UML class notation with a defined set of attributes and methods. The set of methods here can have either: constructors (to construct a VDim) or manipulators (to manipulate the VDim attribute set). Similar to conceptual views, at the XDW conceptual level, VDims can have additional semantic relationships such as generalization, aggregation, association and these can be shown using standard UML notation. In addition to this, two VDims can also have `<<construct>>` relationships with dependencies. Similarly, an *x*FACT can be represented using the `<<xFACT>>`, as shown Fig. 8.2.

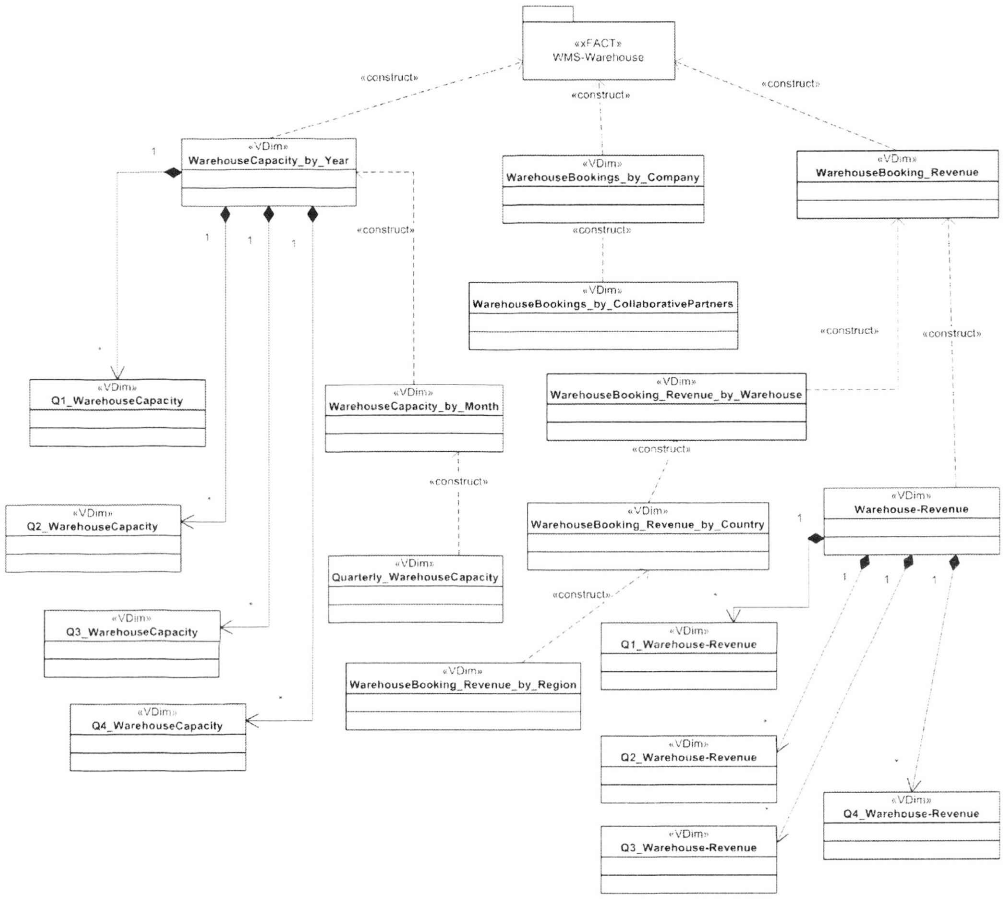


Figure 8.2: A VDim hierarchy in e-Sol-W case study (with UML stereotypes)

We have stated that semantically related conceptual views could be logically grouped together as grouping of classes into a subject area. Further, a new view-hierarchy and/or constructs can be added to include additional semantics for a given user requirement. In the XDW conceptual model, when a collection of similar or related *conceptual views* are logically grouped together, we called it grouped VDims (Fig. 8.2), implying that it satisfies one or more logically related user requirement(s).

In addition, we can also construct additional conceptual view hierarchies as shown in Fig. 8.2. These hierarchies may form additional structural or dependency relationships with existing conceptual views or view hierarchies (grouped VDims) as shown in Fig. 8.2. Thus, it is possible that a cluster of dimensional hierarchy/ies can be used to model a certain set of user requirement/s. Therefore we argue that, this aggregate aspect can give us enough abstraction and flexibility to design a user-centred XDW model.

In order to model a hierarchy of VDim and capture the logical grouping among them, we utilize the ***package*** construct in UML. Thus, a grouped VDim hierarchy as shown in Fig. 8.2 can be represented using the UML package notation (Lujan-Mora, Trujillo & Song 2002a, 2002b; Luján-Mora, Trujillo & Vassiliadis 2004; Vicky Nassis 2006). This in practice describes our logical grouping of conceptual views and their hierarchies. Thus, we utilize packages to model our connected dimensions (Fig. 8.2).

In Fig 8.2, we show our case study XDW model with *x*FACT and VDims connected via the `<<construct>>` stereotype. Also, following the arguments presented, we can show that, the *x*FACT (shown in Fig 8.2) can be grouped into one logical construct and can be shown in UML as one package. A complete *x*FACT model is shown in Fig 8.3.

Example 8.2: In the e-Sol-W example, as shown in Fig 8.2 & 8.3, `WMS_Warehouse_Income` is an *x*FACT.

Example 8.3: Also, in Fig 8.2, a VDim hierarchy is shown using `<<VDim>>`, `<<construct>>` and `<<xFACT>>` stereotypes.

The schemata transformation of *x*FACT and VDim UML specification to the logical model is similar to the schemata transformation described in Chapter 7.

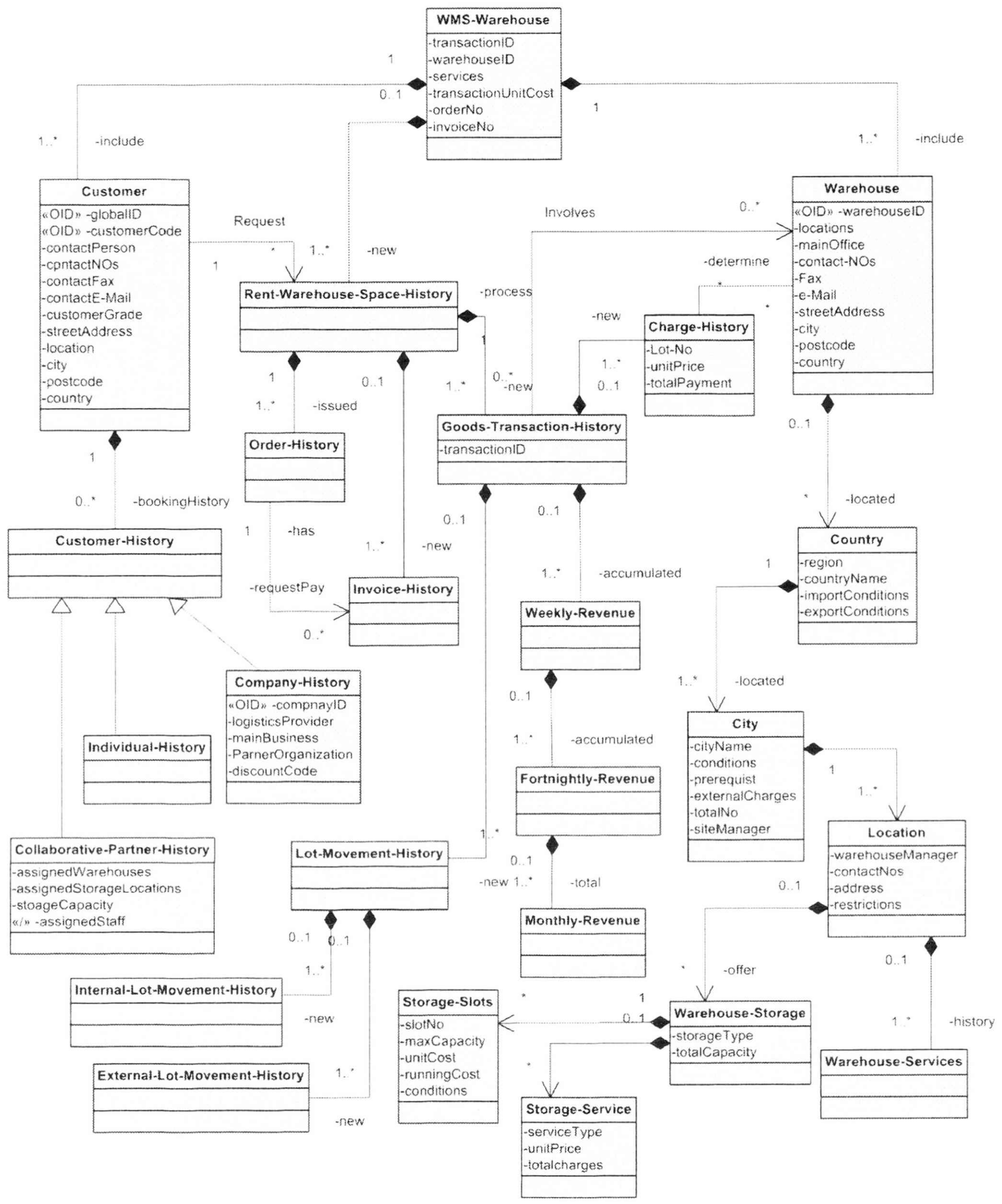


Figure 8.3: An *x*FACT example in the e-Sol-W (WMS-Warehouse)

8.9 View-Driven XML FACT Repository

In this section, we present the $\mathcal{G}$xFACT model, its properties and implementation model options. Intuitively, it can be seen that one can model and design an *x*FACT using conceptual views themselves. That is, *x*FACT may be considered as a collection of hierarchically organized conceptual views. A *conceptual view* driven *x*FACT model is referred to in this research as a Global XML FACT Repository (or $\mathcal{G}$xFACT in short). That is, it is a global (collection) of conceptual views.

At the conceptual level, the $\mathcal{G}$xFACT is analogous to a PIM in the MDA framework. At the logical and document levels (i.e. schema and query expressions with materialized instances), the $\mathcal{G}$xFACT is a PSM. It should be noted that, depending on the deployment platform and the implementation concerns, for a given $\mathcal{G}$xFACT (PIM) one or more alternative logical (and document) level $\mathcal{G}$xFACT (PSM) constructs are possible.

8.9.1 PIM: $\mathcal{G}$xFACT Properties

$\mathcal{G}$xFACT is an *architectural construct* (rather than an implementation or storage structure) to build and integrate multiple operational data sources, XML data marts and/or other XML FACT repositories into one global XML FACT repository that provide perspectives to an organization at a higher level than at the individual business and/or stakeholders' level. The global level as shown in Fig. 8.4, is an aggregated *x*FACT (i.e. *x*FACT of one or more *x*FACTs, data marts, etc.) to support DSS, MIS and EIS solutions in global business environments, in the context of a global XML document warehouse.

The $\mathcal{G}$xFACT model utilizes the layered view model to provide three levels of abstraction, namely conceptual, logical/schema and document/instant levels, which in turn uses the OOCM approach to data warehousing and the industry standard UML as the modelling language.

The design methodology starts from initial (operational) data level to the global repository level, as shown in Fig. 8.4. Since it is based on high-level modelling (conceptual and logical) semantics, it is independent of the implementation platform/storage architecture or the operational data model. Also, it does not matter which implementation model (or platform) is chosen for the $\mathcal{G}$xFACT, as long as the storage model supports native XML data. Thus, we say that the model and design of $\mathcal{G}$xFACT is platform independent (or produces a Platform-Independent-Model (PIM)). Since the basic constructs are views, building hierarchical dimensions (VDim) and virtual view support for OLAP queries are built-in to the model, which can be designed using top-down approaches, based on user requirements and/or performance issues.

The $\mathcal{G}$xFACT repository will:

(i) provide an integrated data source for BI tools for a given context (e.g. global/regional earnings);

(ii) provide data and data semantics to built further (global) dimensions and dimensional hierarchies;

(iii) provide seamless integration of existing data warehouse sources with preserved data semantics;

(iv) preserve conceptual semantics of initial data warehouse environments; and

(v) support and reflect changing warehouse requirements at a higher level of abstraction.

In the next section we present some of the formal properties of the $\mathcal{G}$xFACT followed by some modelling associated with it.

Example 8.4: In e-Sol-W example, a conceptual views "`Warehouse-Q1_revenue_China`", "`Warehouse-Q1_revenue_LA`", "`Warehouse-Q1_ revenue_HK`" etc. constructed in the given context of "`Warehouse_Revenue`". The valid context is given by the e-Sol "*x*FACT_WMS" objects.

Example 8.5: A conceptual view "`Warehouse-Capacity-LA`", "`Warehouse-Capacity-China`", "`Warehouse-Capacity-Australia`" and "`Warehouse-Capacity-HK`", in the given context of "`Warehouse-Capacity_by_season`".

Example 8.6: A conceptual view "`Lot-Master-Charge-History`" in the given context of "Lot-Management", in WMS. Here, at the conceptual level, it can be stated as a *materialized* conceptual view, implying that it is a persistence view during the lifetime of the system.

8.9.2 Formal Semantic of $\mathcal{G}$xFACT

Let $t_1, t_2, \ldots. t_i \ldots.$ (where $1 \le i \le n$) be a finite set of operational document databases (ODB, Fig. 8.4 & 8.5) from which an XDW is built (here e-Sol-W) for an organization (here for e-Sol). Here, we consider t_i (as one logical ODB, as one ODB can be physically stored, accessed and/or operational in multiple servers, departments and states/countries) in our e-Sol example.

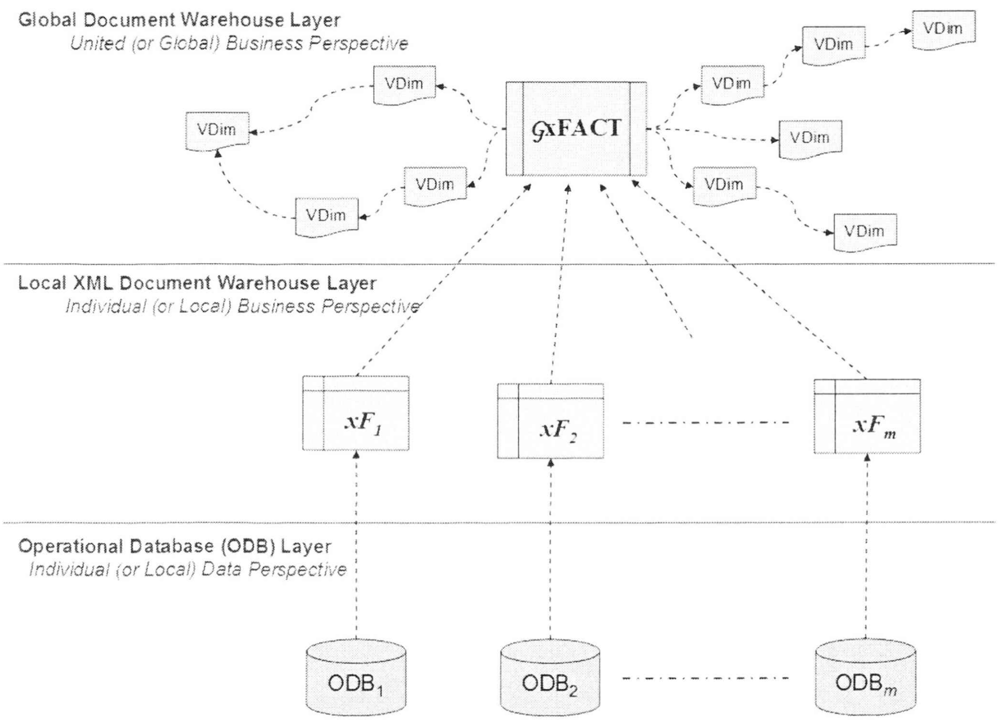

Global Document Warehouse Layer
United (or Global) Business Perspective
VDim
$\mathcal{G}$xFACT
Local XML Document Warehouse Layer
Individual (or Local) Business Perspective
xF_1
xF_2
xF_m
Operational Database (ODB) Layer
Individual (or Local) Data Perspective
ODB_1
ODB_2
ODB_m

Figure 8.4: $\mathcal{G}$xFACT context diagram

Let T be the set that holds all ODBs (one T per XDW, as shown in Fig. 8.4). Thus, $t_i \in T$, where $1 \le i \le n$. Therefore the domain of T (thus domain of the resulting XDW) can be shown as:

$$dom(T) = \bigcup_{i=1}^{n} dom(t_i) \qquad (8.5)$$

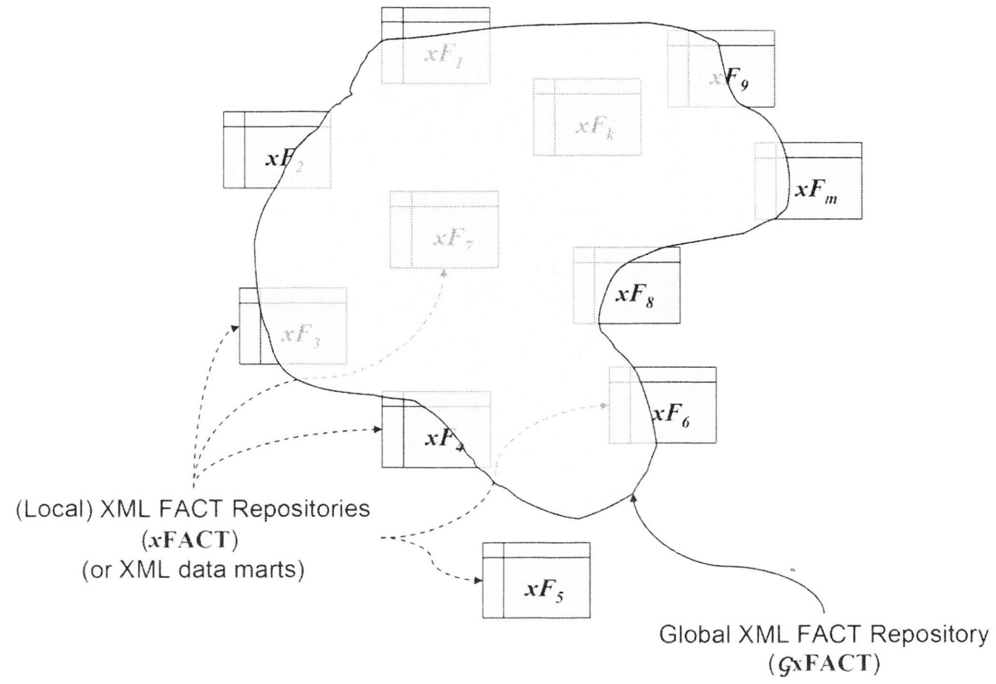


Figure 8.5: *x*FACTs and the view *contexts* (GxFACT)

Let $xFACT$ denote an *x*FACT and $(xF_1, xF_2, ...xF_k.....) \in xFACT$ be a finite set of *x*FACT repositories where $1 \leq k \leq m$ of logically connected, standalone XDWs (XDW of business stokeholds/partners in e-Sol). Based on the conceptual view definition in Chapter 5, let V^c denote a conceptual view that can be defined over a context ζ_{ontext}. From conceptual view properties, it can be shown that V^c can be either:

$$V^c = \zeta_{ontext} \text{ or } V^c \subseteq \zeta_{ontext} \qquad (8.6)$$

Let us consider an *x*FACT, xF_k (Fig. 8.4 – 8.5), as a specialized context (whole *x*FACT/(s) or partial, as shown as the *highlighted* region in Fig. 8.5) that is of interest to the organization at the global level. Let v_k, ($v_k \in V^c$) be a conceptual view defined over the *x*FACT xF_k, where v_k represent the items in xF_k that is of some interest to the organization at the global level. This can be shown as:

$$xF_k \cap GxFACT = v_k \qquad \textbf{(8.7)}$$

From equation 8.6, it can be shown that, v_k can either be $v_k = xF_k$ or $v_k \subseteq xF_k$. This v_k is shown in Fig. 8.5 as the area of intersection between an *x*FACTs (from xF_1 to xF_m) and $\mathcal{G}$xFACT (highlighted or shaded area). Therefore, it can be shown that the resulting $\mathcal{G}$xFACT is union of all such conceptual views v_k. This is shown below:

$$GxFACT \supseteq \bigcup_{k=1}^{m} v_k \qquad \textbf{(8.8)}$$

Also, in this case, since the domain of v_k is either $dom(v_k) = dom(xF_k)$ or $dom(v_k) \subseteq dom(xF_k)$. Also, it can be shown that the, domain of the resulting $\mathcal{G}$xFACT:

$$dom(GxFACT) \supseteq \bigcup_{k=1}^{m} v_k \qquad \textbf{(8.9)}$$

Let $vd_j^p \in VD^l$ ($p := 1 \rightarrow m$ and $j \geq 1$)where, VD^p is a finite set of *virtual dimensions* (VDim), defined over a given *x*FACT repository $xF_l \in xFACT$. Similarly, based on the work, it can be shown that there can be many VDim (or VDim hierarchy/(ies)) defined over the $\mathcal{G}$xFACT to satisfy the global information need of an organization.

Thus, formally, since $\mathcal{G}$xFACT is a semantically related hierarchical collection of conceptual views, at the conceptual level, we can define a $\mathcal{G}$xFACT as:

Definition 8.4. A global XML FACT repository (*G*xFACT) GxF^c is a 4-ary tuple with GxF^c_{name}, GxF^c_{obj}, GxF^c_{rel} and $GxF^c_{constraint}$, where, GxF^c_{name} is the name of the *G*xFACT repository GxF^c, GxF^c_{obj} is a set of objects in GxF^c, GxF^c_{rel} is a set of object relationships in GxF^c, and $GxF^c_{constraint}$ is a set of constraints associated with GxF^c_{obj} and GxF^c_{rel} in GxF^c.

$$GxF^c = (GxF^c_{name}, GxF^c_{obj}, GxF^c_{rel}, GxF^c_{constraint}) \qquad \textbf{(8.10)}$$

If $V^c_{XDW_i}$ denote a set of conceptual views (say, the i^{th} conceptual view constructed from the i^{th} XML FACT repository is $V^c_{XDW_i}$) define the *G*xFACT from the underlying XML FACT repositories (Fig. 8.5), GxF^c can be stated as:

$$GxF^c \supseteq \bigcup_{i=1}^{n} V^c_{XDW_i} \qquad \textbf{(8.11)}$$

where, n is the finite number of XML FACT repositories from which the GxF^c is the *G*xFACT constructed.

Example 8.7: In our e-Sol example, for a *G*xFACT "`Warehouse-Capacity`", many VDim's such as "`Regional-Warehouse-Capacity-by-Season`", "`Warehouse-Capacity-by-Country`", "`Warehouse-Availability-for-Logistics`", "`Profit-by-Region`" etc. can be defined, providing regional and/or global perspectives.

Example 8.8: In our e-Sol example, in addition to XDWs for local warehouse owners, regional *G*xFACT (e.g. Europe, USA and China) can be built to support growing business demand and/or if a requirement exists, where warehouse/logistics turn-over is very high. For example, "`US-Logistics-Orders`".

8.9.3 Design Steps in Modelling $\mathcal{G}$xFACT

Here we briefly show the steps involved in building a $\mathcal{G}$xFACT model.

- ***Step 1***: Individual XDW models (*x*FACTs and associated VDims) conceptually designed and implemented for each subject-group and/or organization unit.

Here, priorities are given to capture local business needs and requirements (such as a local conference committee and/or host institutions) to support their business (publishing and cataloguing) at the local (not global) business entity level. The *x*FACT and the VDims designed reflect individual and local business needs. Since the VDims are constructed using views, they can readily reflect the changing local business paradigm. This step is comparable to traditional data warehouse engineering, except the requirement engineering methodology is based on XML and the warehouse architecture is based on XDW.

- ***Step 2***: Global XDW requirement is developed (using the requirement engineering technique in) based on all existing XDW (mainly *x*FACT conceptual models) and the available data.

Here, the main goal is to treat all participating business stakeholders (libraries, online catalogue providers, universities, etc.) as one business entity. The output of this step is to amalgamate global business requirements into a requirement model for developing a global XDW.

- ***Step 3***: Correlated XDW (mainly *x*FACT) models collected and crossed-checked for duplicity and ambiguity in the context of the global XDW model.

- ***Step 4:*** Multiple contexts (Fig 8.5) are developed over existing (or new) *x*FACT repositories, data sources, etc. of the local stakeholders.

- ***Step 5:*** For each correlated XDW, conceptual views are defined over their *x*FACT repository models to define elements of interest for the global XDW.

- ***Step 6:*** Based on the requirement model (defined in Step 2) and contexts (and conceptual views) defined in Sept 3-5 above, $\mathcal{G}$xFACT conceptual model is developed. Fig. 8.4 – 8.5 shows these as context diagrams.

- ***Step 7:*** Based on the conceptual model of $\mathcal{G}$xFACT, (collection of conceptual views), logical model (i.e. equivalent logical views and schemas for the conceptual views) is developed for the $\mathcal{G}$xFACT.

Such transformations (i.e. conceptual to logical views) are discussed in Chapters 5, 6 and 7 two OO modelling languages such as XSemantic net and UML.

- ***Step 8:*** The $\mathcal{G}$xFACT model operators (conceptual operators) are transformed into query constructs.

- ***Step 9:*** The $\mathcal{G}$xFACT logical model is implemented. That is, all $\mathcal{G}$xFACT schemas (underlying conceptual view schemas) are made persistent and online.

- ***Step 10:*** Depending on the implementation solution adopted, the $\mathcal{G}$xFACT data is instantiated.

Note: this step is comparable to the relational data warehouse ELT process, which itself is a detailed process and not discussed in detail here. This step includes many operations such as:

(i) data cleaning,

(ii) data caching,

(iii) data merging,

(iv) data aggregation,

(v) data mapping, and data loading to name a few.

8.9.4 PSM: Implementation Choices for the $\mathcal{G}$xFACT

In the $\mathcal{G}$xFACT model: (i) the data source for the instance data comes from one or more *x*FACT of either XDW or XML document marts; (ii) the Meta-Data (due to the nature of XML/XML Schema) is embedded within the source data; and (iii) the $\mathcal{G}$xFACT is designed using (conceptual and logical) views. Due to these factors, we provide three implementation solution models to implement the Global XDW (namely the $\mathcal{G}$xFACT and the associated VDims).

- ***Option 1***: Fully *persistent* (or materialized) $\mathcal{G}$xFACT repository with dimensions. In this option, the $\mathcal{G}$xFACTs (the collections of views that form the $\mathcal{G}$xFACT) are fully materialized. New VDims can be defined over this repository as in the simple XDW model. Also, if needed, depending on user requirements and performance, some VDims can be materialized (e.g. to support OLAP queries).

This option is preferred in a situation where dimensional definitions are dynamic in nature (user query, to support third-part analytical tool etc.), and high-performance computing power (and network resources) are in abundance. This also suits well where the $\mathcal{G}$xFACT should remain reasonably static (e.g. due to the underlying XDW data sources and connectivity) and updated constantly in regular intervals to maintain data accuracy (Mohania, Karlapalem & Kambayashi 1999).

- ***Option 2***: Non-persistence (or non-materialized) $\mathcal{G}$xFACT repository with persistence global dimensions. This option is unorthodox, where the $\mathcal{G}$xFACT logical model is implemented (i.e. schemas, environment parameters etc.), but data is not materialized at the $\mathcal{G}$xFACT level. This situation is comparable to a view definition stored in a relational model. But all the VDims are defined and materialized with their data. Here, $\mathcal{G}$xFACT serves as a meta-data repository rather than a XDW repository.

This option is preferred when an organization has fixed warehouse requirements (at the global level) and a wide-range of high-performance storage solutions (not computational power, such as grid or cluster computing). Also, this solution is feasible if the underlying operational data sources are updated over a longer term than in regular, short (weekly or monthly) intervals. Another advantage of this option is that, since all the dimensions are already materialized (i.e. all complicated query processing is done and data is readily available for end-users), end-users do not require high-performance computing power; thus, it is suited well for regions that suffer from such issues.

Option 3: Here, it is a combination of options 1 and 2, where predefined sections of the $\mathcal{G}$xFACT repository (i.e. selected views) and selected VDims (i.e. dimensional views) are materialized based on business (and performance) requirements.

In making a decision on which option to choose, the following factors should be considered:

- $\mathcal{G}$xFACT requirements,
- availability of computing power (and associated resources),
- end-user computing resources,
- end-user knowledge,
- support for in-house/third party analytical tools,
- estimated size (and predicted growth rate) of the $\mathcal{G}$xFACT and

- required performance level.

For example, if a section of the business or regional XDW (or data availability) suffers from lack of computing and network resources, $\mathcal{G}$xFACT/VDim sections associated with such data may be materialized to improve data availability and/or performance for the overall global XDW.

Note: In deciding on option 3, in addition to warehouse and business requirements (steps 1-3), warehouse operational requirements must be considered (which section/(s) and/or VDim to materialize etc.) in designing the $\mathcal{G}$xFACT conceptual model.

8.10 Conclusion

In this chapter, we presented an intuitive approach to model and design an XDW model, using the views in the LVM. First, we presented some background to the concept of data warehouse and XML, followed by some related work done in the area of conceptual models for data warehouses. Then, we described our own proposed model, the XML Document Warehouse model, its unique characteristics and the four design levels. Thirdly, we presented some formal semantics and definitions associated the XDW conceptual model. Later, we describe the modelling and transformation issues of the XDW conceptual model.

Finally, we described an intuitive approach to designing a view-driven *x*FACT repository, its properties, formal semantics and modelling and implementation options. We presented an intuitive (three-layered), view-driven, *architectural construct* to conceptually model, design and implement a global XML FACT repository, $\mathcal{G}$xFACT. Such a repository will unite organizations' existing warehouse solutions (XML Document Warehouses and XML marts) to provide an integrated (or global) perspective for DSS, MIS and EIS.

Since all the concepts presented in this chapter are analogues to the LVM, only a brief discussion of the schemata transformation was presented. The Chapters 6 and 7 provide a detailed discussion of this transformation.

In the following two chapters, we will continue to describe two more case studies and applications of the LVM in the context of: Semantic Web and ontology (Chapter 9) and website and web portal design (Chapter 10).

References

Abelló, A, Samos, J & Saltor, F 2001, 'Understanding facts in a multidimensional object-oriented model', *4th International Workshop on Data Warehousing and OLAP (DOLAP '01)*.

Cluet, S, Veltri, P & Vodislav, D 2001, 'Views in a Large Scale XML Repository', *Proceedings of the 27th VLDB Conference (VLDB '01)*, Roma, Italy.

Debevoise, T 1999, *The Data Warehouse Method: Integrated Data Warehouse Support Environments*, Prentice Hall PTR, USA.

Dillon, TS & Tan, PL 1993, *Object-Oriented Conceptual Modeling*, Prentice Hall, Australia.

ECM-AIIM 2005, *The ECM Association (http://www.aiim.org/index.asp)*, AIIM, http://www.aiim.org.

Elmasri, R & Navathe, S 2004, *Fundamentals of database systems*, 4th edn, Pearson/Addison Wesley, New York.

Elmasri, R & Navathe, SB 2000, *Fundamentals of database systems*, 3 edn, Addison-Wesley, Reading, Mass. Harlow.

Fankhauser, P & Klement, T 2003, 'XML for Data Warehousing Changes & Challenges', *DaWaK 2003*, Springer, Prague, pp. 1-3.

Feng, L & Dillon, TS 2003, 'Using Fuzzy Linguistic Representations to Provide Explanatory Semantics for Data Warehouses', *IEEE Transactions on Knowledge and Data Engineering (TOKDE)*, vol. 15, No. 1, no. 1, pp. 86-102

Giovinazzo, WA 2000, *Object-oriented data warehouse design : building a star schema*, Prentice Hall PTR, Prentice-Hall International, Upper Saddle River, N.J. London.

Golfarelli, M, Rizzi, S & Vrdoljak, B 2001, 'Data warehouse design from XML sources', *Proceedings of the 4th ACM international workshop on Data warehousing and OLAP*, ACM Press, Atlanta, Georgia, USA, pp. 40 - 7.

Gopalkrishnan, V, Li, Q & Karlapalem, K 1999, 'Star/Snow-flake Schema Driven Object-Relational Data Warehouse Design and Query Processing Strategies', *1st First International Conference on Data Warehousing and Knowledge Discovery (DaWaK '99)*, Springer, Florence Italy.

Gray, P & Watson, HJ 1998, *Decision Support in The Data Warehouse*, Prentice Hall PTR, USA.

Gupta, A, Mumick, IS & (eds) 1999, *Materialized views: techniques, implementations, and applications*, eds A Gupta & IS Mumick, MIT Press.

Humphries, M, Hawkins, MW & Dy, MC 1999, *Data Warehousing: Architecture & Implementation*, Prentice Hall PTR, USA.

Inmon, WH, Imhoff, C & Battas, G 1996, *Building the operational data store*, John Wiley & Sons, New York, USA.

Jacek Blazewicz, Wieslaw Kubiak, Tadeusz Morzy, Rusinkiewicz, M & (eds) 2003, *Handbook on Data Management in Information Systems*, eds Jacek Blazewicz, Wieslaw Kubiak, Tadeusz Morzy & MR (eds), Springer, Berlin ; New York.

Jeong, E & Hsu, C-N 2001, 'Introduction of Integrated View for XML Data with Heterogenous DTDs', *Proceedings of the 10th International Conference on Information and Knowledge Management (CIKM '01)*, ACM, Atlanta, Georgia, pp. 151-8.

Johnson, T 2002, 'Data warehousing', in *Handbook of massive data sets*, Kluwer Academic Publishers, pp. 661-710.

Kimball, R & Caserta, J 2004, *The data warehouse ETL toolkit : practical techniques for extracting, cleaning, conforming, and delivering data*, Wiley, Hoboken, NJ.

Kimball, R & Ross, M 2002, *The data warehouse toolkit : the complete guide to dimensional modeling*, 2nd edn, Wiley, New York.

Lucie-Xyleme 2001, 'Xyleme: A Dynamic Warehouse for XML Data of the Web', *International Database Engineering & Applications Symposium (IDEAS '01)*, eds ME Adiba, C Collet & BC Desai, IEEE Computer Society 2001, Grenoble, France, pp. 3-7.

Luján-Mora, S & Trujillo, J 2004, 'A Data Warehouse Engineering Process', *Third International Conference on Advances in Information Systems (ADVIS '04)*, Springer-Verlag GmbH, Izmir, Turkey, p. 14.

Lujan-Mora, S, Trujillo, J & Song, I-Y 2002a, 'Extending the UML for Multidimensional Modeling', *Fifth International Conference on the Unified Modeling Language and its applications (UML '02)*, Springer-Verlag London, UK, Dresden, Germany, pp. 290-304.

Lujan-Mora, S, Trujillo, J & Song, I-Y 2002b, 'Multidimensional Modeling with UML Package Diagrams', *Proceedings of the 21st International Conference on Conceptual Modeling (ER '02)*, Springer-Verlag London, UK, pp. 199-213.

Luján-Mora, S, Trujillo, J & Vassiliadis, P 2004, 'Advantages of UML for Multidimensional Modeling', *Proceedings of the 6th International Conference on Enterprise Information Systems (ICEIS '04)*, Porto, Portugal, April.

Luján-Mora, S, Vassiliadis, P & Trujillo, J 2004, 'Data Mapping Diagrams for Data Warehouse Design with UML', *23rd International Conference on Conceptual Modeling (ER '04)*, vol. 3288, Springer-Verlag GmbH, Shanghai, China, p. 191.

Medina, E, Luján-Mora, S & Trujillo, J 2002, 'Handling Conceptual Multidimensional Models Using XML through DTDs', *Proceedings of the 19th British National Conference on Databases: Advances in Databases*, vol. 2405, Springer, pp. 66 - 9.

Mohammed, S 2001, 'Object-Relational Data Warehouse', Master by Coursework (Major Thesis) thesis, La Trobe University, Melbourne, Australia.

Mohania, MK, Karlapalem, K & Kambayashi, Y 1999, 'Data Warehouse Design and Maintenance through View Normalization', *10th International Conference on Database and Expert Systems Applications (DEXA '99)*, vol. 1677, Springer, Florence, Italy, pp. 747-50.

Nassis, V 2006, 'Requirements Based and Conceptual Approach for XML Document Warehouse Analysis and Design', Doctor of Philosophy (Ph.D) thesis, La Trobe University, Australia, Melbourne (to be submitted).

Nassis, V, Dillon, TS, R.Rajugan & Rahayu, W 2006 - *to appear*, 'An XML Document Warehouse Model', *The 11th International Conference on Database Systems for Advanced Applications (DASFAA '06)*, Springer, Singapore.

Nassis, V, Dillon, TS, Rahayu, W & R.Rajugan 2006, 'Goal-Oriented Requirement Engineering for XML Document Warehouses', in J Darmont & O Boussaid (eds), *Processing and Managing Complex Data for Decision Support*, Idea Group Publishing, pp. 28-62.

Nassis, V, R.Rajugan, Dillon, TS & Rahayu, JW 2005a, 'A Requirement Engineering Approach for Designing XML-View Driven, XML Document Warehouses', *The 29th Annual International Computer Software and Applications Conference (COMPSAC '05)*, IEEE Computer Society, Edinburgh, Scotland, pp. 388-95.

Nassis, V, R.Rajugan, Dillon, TS & Rahayu, W 2005b, 'Conceptual and Systematic Design Approach for XML Document Warehouses', *International Journal of Data Warehousing and Mining*, vol. 1(3), no. 3, pp. 63-87

OMG-CWM 2001, *The Common Warehouse Metamodel (CWM™) (http://www.omg.org/technology/cwm/)*, OMG, 2005.

OMG-MDA 2003, *The Architecture of Choice for a Changing World®, MDA Guide Version 1.0.1 (http://www.omg.org/mda/)*, OMG, 2005.

OMG-MOF™ 2003, *Meta-Object Facility (MOF™), Ver 1.4 (http://www.omg.org/technology/documents/modeling_spec_catalog.htm#MOF)*, OMG, 2005.

Pokorn'y, J 2002, 'XML Data Warehouse: Modelling and Querying', *Proceedings of the Baltic Conference (BalticDB-IS '02)*, vol. 1, eds H-M Haav & A Kalja, Institute of Cybernetics at Tallin Technical University.

Ponniah, P 2001, *Data Warehousing Fundamentals: A Comprehensive Guide for IT professionals*, John Wiley & Sons Inc., NY.

Rahayu, W, Dillon, TS, Mohammed, S & Taniar, D 2001, 'Object-Relational Star Schemas', *13th IASTED International conference on Parallel & Disibuted Computing and Systems (PDCS '01)*, IASTED, LA, USA.

Theodoratos, D & Sellis, T 1999, 'Dynamic Data Warehouse Design', *1st International Conference on Data Warehousing & Knowledge Discovery (DaWak '99)*, Springer, Italy, pp. 1-10.

Trujillo, J & Luján-Mora, S 2003, 'A UML Based Approach for Modeling ETL Processes in Data Warehouses', *Int. Conf on Conceptual Modeling (ER '03)*, Springer-Verlag GmbH, pp. 307-20.

Trujillo, J, Luján-Mora, S & Song, I-Y 2003, 'Applying UML For Designing Multidimensional Databases And OLAP Applications. 13-36', *Advanced Topics in Database Research, Idea Group Publication*, vol. 2, pp. 13-36

Trujillo, J, Palomar, M, Gomez, J & Song, I-Y 2001, 'Designing Data Warehouses with OO Conceptual Models', *IEEE Computer Society, "Computer"*, December, 2001, pp. 66-75.

W3C-XML 2004, *Extensible Markup Language (XML) 1.0, (http://www.w3.org/XML/)*, The World Wide Web Consortium (W3C).

Xyleme 2001, *Xyleme Project (http://www.xyleme.com/)*, http://www.xyleme.com/.

Chapter 9

Case Study & Applications II: Views for Semantic Web, View-Driven Ontology Extraction

9.1 Introduction

In the previous chapter, a conceptual framework for modelling and designing an XML Document Warehouse (XDW) was presented. In this chapter, we present our research results in extending the LVM for the Semantic Web (SW) paradigm (W3C-SW 2005). A further continued discussion of the research described in this chapter is also proved in Chapter 11 as part of the proposed future research directions.

As presented in Chapter 2, the notion of view in the Semantic Web paradigm is a relatively new research direction. To date, there exist some view formalisms that are proposed in works such as (Volz, Oberle & Studer 2003; Wouters 2006; Wouters et al. 2004b). However, except for the work by Wouters et al. in (Wouters 2006), most of these view formalisms focus on providing a view formalism (mostly based on a SW languages such as RDF (W3C-RDF 2004), OWL(W3C-OWL 2004) etc. Only the work in (Wouters 2006), provides a declarative notion of conceptual modelling of ontologies and sub-ontology extraction within this context, a conceptual notion of views was considered.

In our research in the LVM for XML, in Chapter 4, we presented arguments for having the notion of views at a higher level of abstraction that are independent of implementation (i.e. queries, data manipulation languages) and schematic concerns. Here, these arguments are still valid in the case of the SW paradigm. Thus, in this chapter, we look into proposing a LVM like conceptual framework for the SW paradigm.

This chapter is organized as follows. In section 9.2, we present a general introduction to SW and SW concepts. This is followed by some background information on some of the concepts and terminologies that are different from the traditional data domain in section 9.3. Section 9.4 provides an extended discussion of the motivation for our research in this chapter, the ontology views, followed by section 9.5 that elaborates on our e-Sol case that is used here to illustrate our concepts. In section 9.6, we define and provide formal semantics to some of the concepts and notions used in this chapter. This is followed by section 9.7, where a brief discussion of ongoing work on one of the proposed deployment platforms for our ontology views. Section 9.8 concludes this chapter. It should be noted here that some extended discussion of our ongoing research in regards to SW paradigm and ontology views are also discussed in Chapter 11.

9.2 The Semantic Web

The World Wide Web (WWW) is widespread and constantly growing with vast amounts of information being disseminated using a variety of technologies available to-date. But, unlike databases, it has created the problem of *standardization* vs. *customization*, where anyone can represent what s/he wants according to one's own representation and annotations. Thus, the traditional web is not semantically rich, and ad-hoc queries are usually done using search engines that query the web using strings rather than search by the meaning of data. This has triggered a series of new research directions such as metadata description languages, search engines (Luk et al. 2002), and knowledge representation (Aberer et al. 2004) techniques for the web to organize, describe, manage, disperse and query information in an orderly manner using the web as the new medium. To overcome some of problems associated with these issues, researchers have recently proposed the Semantic Web (SW) approach.

The Semantic Web, envisioned by Berners-Lee (Berners-Lee 1999), promised to make the web a meaningful experience. SW enables users to define and represent information structures, using widely accepted metadata standards to annotate new vocabularies that are independent of any language or syntax used to implement it. These accepted metadata standards are formulated into an *ontology*, a notion which is the foundation of the SW and one of the keys to the success of SW.

9.2.1 Ontology

An ontology is the notion of *shared conceptualization* (Gruber 1993). It can also be stated as a specification of a conceptualization of a problem domain (Spyns, Meersman & Mustafa 2002; Wouters et al. 2004b) or may be viewed as a *shared conceptualization* of a domain that is commonly agreed to by all parties (Chang, Dillon & Hussain 2005). Formally, an ontology may be stated as "a specification of a Conceptualization" (Gruber 1993). Here, "conceptualization" refers to the understanding of the *concepts* and the *relationships* between the concepts that may exist or do exist in a specific domain or a community (Chang, Dillon & Hussain 2005). Also, a specification of the conceptualization refers to the notion or the representation of the commonly agreed, domain specific shared knowledge. There are many other ways of expressing ontologies, but such simplified definitions may incorrectly lead to oversimplification of the ontology concept, i.e. ontology is a collection of words, definitions and concepts, a similar notion of knowledge bases in the classical artificial intelligence communities. Therefore, it should be noted that an ontology is more than a knowledge base, and not merely a collection of distributed metadata repositories, or annotated traditional databases. A brief outline highlighting the differences between various traditional database concepts and ontology concepts is given in Section 9.3 below.

During the first phase of the SW adoption, many research directions have been focused on formulating ideas and concepts to model and define ontology representation standards. During the second phase (present day) (Wouters et al. 2004b), issues addressed include:

(i) theoretical and practical foundation for modeling ontology bases,

(ii) efficient techniques for querying, integrating, versioning, distribution and maintaining large-scale ontology bases,

(iii) re-integration of existing systems and

(iv) framework for developing semantic web applications.

In the context of SW, theoretically, the entire world is modelled as one super-ontology (Spyns, Meersman & Mustafa 2002; Wouters et al. 2004b), providing great compatibility and consistency across all sub-domains. This illustrates the known characteristics of an ontology, where, in any application domain, an ontology tends to grow larger than its original size, introducing problems such as:

(i) being too large to be utilized in its original scale by potential applications; and

(ii) capable of being understood by the community due to its size.

Another interesting characteristic that is worth mentioning here is the notion of *generic* vs. *specific* ontologies. This is one of the key features that distinguish ontology bases from traditional data repositories. Given a domain, it can be represented using both generic (and upper) ontologies and specific (or lower) ontologies. Simply stated, a specific ontology commits to the use of all generic ontology concepts and specification (Chang, Dillon & Hussain 2005). The specific ontologies are also known as *ontology commitments* (Spyns, Meersman & Mustafa 2002). The specific ontology may involve additional constraints and additional elaborations (Chang, Dillon & Hussain 2006). However, in this research we do not elaborate on this feature of ontologies, as the detailed discussion on ontology and ontology classification is outside the scope of this thesis. In the following sections, we look at some of the concepts that are relevant to our research and to this chapter

9.2.2 Sub-Ontologies

As described before, an ontology base tends to grow larger with usage. Therefore, there exists an interest to extract a portion (i.e. sub-section or sub-ontology) of the ontology that is of interest to the user for a given task. This would have benefited both users from over-utilizing valuable computing resources and time in carrying out their tasks, and would have provided other benefits such as privacy, security and efficient access. Another reason for localized sub-ontology is the processing time involved executing semantic queries over large-scale ontology bases (for example medical ontology UMLS). In recent years, many academic and industrial research directions have been focused on these issues and some of the notable works include (Bhatt et al. 2004; Do & Rahm 2004; Noy & Musen 2002; Volz, Oberle & Studer 2003; Wouters et al. 2002; Wouters et al. 2004a).

For example, if we take the human disease classification (HDC) (Hadzic & Chang 2004) ontology base or medical ontology (UMLS), at this point in time, only a sub-section of the vast ontological structure is required by a medical analyst (or doctor) to complete his/her task. But due to its size, HDC (or UMLS) in its entirety is a drawback as it presents one with a vast amount of un-wanted vocabularies for a given task, thus slowing or preventing the user from making time-critical decisions. Also, given the vast amount of classified data (such as medication, patient history etc.) in the HDC, providing privacy, security and access to all the information present in the HDC for number of users, is an unmanageable task. But, given a sub-ontology (Wouters 2006) extracted from the base ontology for a given user profile, it is a manageable task. A sub-ontology is not just a portion of the existing ontology (Wouters 2006). Rather, it is a view which may involve keeping the required nodes and concatenations of the relationships though nodes have been removed.

However, there exist many issues that need addressing in the case of ontology views (i.e. sub-ontologies).

- First, unlike database or semi-structured views (Wouters 2006; Wouters, Dillon et al. 2005; Wouters et al. 2006), sub-ontologies are not just an extracted portion of the main ontology base, but a collection of concepts, relationships and concerns that itself is a new interpretation of the base ontology.
- Second is the meaningful representation of such sub-ontologies that are easily understood by humans as well as easily transformed to machine- (or user-application) readable notations that are at a level of abstraction that is capable of interpreting, querying and processing of concepts, relationships and constraints.
- Third, are the complementary issues associated with sub-ontologies (Gruber 1993; Noy & Musen 2002; Wouters et al. 2006), such as view

maintenance, versioning and materialization, that are synonymous with database views, but deserve detailed studies of their own, in the context of ontologies.

- Fourth is the issue of meaningful, yet efficient extraction of sub-ontologies from distributed base ontologies (Bhatt et al. 2004; Wouters et al. 2004b).

In the next section, in addition to our discussion in Chapter 2, we present further background information for this chapter, including some discussion of the relationship (or absence of such relationships) and the differences between SW and database concepts.

9.3 Background

In this section, as a supplement to the discussion presented in Chapter 2 (on views for Semantic Web, Section 2.6) we elaborate on some of the common terminologies, differences and concepts between the SW paradigm and traditional databases. Databases and ontologies serve to structure a vast amount of information that is available (Wouters 2006) at a given point in time. But in theory, there exists a clear distinction between databases and ontologies, namely, the clear distinction between the schema and the instances for each of theses. In databases (relational, OO, active, etc.), schemas are precisely defined in one layer of abstraction (usually at the logical or schemata level) and instances are added, edited and/or validated in another layer. Usually, views in database systems are defined as part of the external schema. Conversely, Ontology bases tend to have heterogeneous schemas at varying levels of abstraction (logical or instantiated schemas) and instances may co-exist among these schemas to convey information, concepts or relationships between two concepts to the users.

Another intriguing difference between database and an ontology base is that, a database tends to follow a well-defined and established standard/(s), while ontology standards, functionality and definitions tend to differ between implementations and models due to its infancy (Wouters et al. 2002; Wouters et al. 2004b). For example, in

OWL (W3C-OWL 2002) one can create instances as part of the ontology but not in the DOGMA approach (Spyns, Meersman & Mustafa 2002).

For the purpose of this research, we need to make a distinction between the concept of high-level view definitions (such as definitions in the LVM for XML) for SW and the view definitions in SW languages such as Resource Description Framework (RDF) (W3C-RDF 2004) and the Ontology Web Language (OWL) (W3C-OWL 2002). Though expressive, SW related technologies and languages suffer from visual modelling techniques, fixed models/schemas and evolving standards. In contrast, higher-level OO modelling language standards (with added semantics to capture Ontology domain specific constraints) are well-defined, practised and transparent to any underlying model, language syntax and/or structure (Cruz et al. 2002; Gâomez-Pâerez, Fernâandez-Lâopez & Corcho 2003). They also can provide well-defined models that can be transferred to the underlying implementation models with ease. Therefore, for the purpose of this paper, an abstract view for SW is a view, where its definitions are captured at a higher level of abstraction (namely, conceptual), which in turn can be transformed, mapped and/or materialized at any given level of abstraction (logical, instance etc.) in a SW specific language and/or model.

In addition, an abstract view model for SW should be able to deal with not just one, but multiple, data encoding language standards and schemas (such as XML, RDF, OWL etc.), as enterprise content may have not one, but multiple data coding standards and ontology bases. Another issue that deserves investigation is the modelling techniques of views for SW. Though expressive, SW related technologies suffer from insufficient visual modelling techniques (Cruz et al. 2002). This is because Object-Oriented (OO) modelling languages (such as UML) provide insufficient modelling constructs for utilizing semi-structured (such as XML, RDF, OWL) schema based data descriptions and constraints, while XML/RDF Schema lacks the ability to provide higher levels of abstraction (such as conceptual models) that are easily understood by humans. But many researchers have proposed OMG's UML (OMG-UML™ 2003b) (and OCL) based solutions (Cruz et al. 2002; Dragan Gaševic et al. 2004a, 2004b; D. Gaševic et al. 2004; Wouters et al. 2002; Wouters et al. 2004b), with added extensions to model semi-structured data.

9.4 Motivation

The emergence of SW and related technologies promise to make the web a meaningful experience. Today, there is an increased awareness of SW and ontology applications in domains of traditional enterprise functions and services over the web, in a secure controlled environment. However, high level modelling, design and querying techniques prove to be a challenging task for organizations that are hoping to utilize the SW paradigm for their industrial applications. Conversely, the SW paradigm can benefit, if some of the traditional database concepts, functionalities and models are made adoptable to the new SW paradigm.

This creates the need to investigate successful database technologies, such as views, in the context of SW, where ontology views (Wouters et al. 2004b) can be used for:

(i) Ontology extraction;

(ii) Ontology versioning;

(iii) SW-enabling traditional data sources;

(iv) Sub-ontology extraction, in an industrial setting; and

(v) Applications of SW.

However, as described before, unlike traditional database systems, high level modelling, design and querying techniques still prove to be challenging for the SW paradigm. This is mainly due to the nature of ontology bases and views, where, definitions and querying have to be done at the high-level abstraction (Volz, Oberle & Studer 2003; Wouters et al. 2004b). Such high-level view models can also be utilized in the SW paradigm and also support and co-exist with existing traditional database architectures and/or enterprise transactional systems.

Given the benefits and concerns of sub-ontologies, the motivation for this research consists of several related sub-problems and these are:

(i) the development of an approach that allows the separation of *implementation* aspects and the *conceptual* aspects of the ontology views so that there is a clear separation of concerns allowing analysis and design of ontology views to be separated from their implementation (i.e. view extraction and elaboration);

(ii) the development of semantics for ontology views as described in point (i) as analogous to LVM;

(iii) to achieve point (i) above, requires extensions and elaborations of the conceptual operators presented earlier, in the context of LVM for XML, to permit extraction and elaboration of construction of ontology views at a higher level of abstraction;

(iv) to define representations to express these ontology views at the conceptual level analogous to the LVM. This requires specification of customized graphical notations and the development of automated schemata transformation rules for mapping conceptual models to OWL (or RDF-S) like representations;

(v) to investigate feasible deployment strategies for the proposed ontology views.

(vi) to define a view development methodology for *ontology views* that utilizes the conceptual models and carries out automated transformation to a *template* (such as in RDF-S) for the views as well as a representation of the view construction in an appropriate query language such as XQuery (W3C-XQuery 2004), RDQL, etc.; and

(vii) to apply the concepts, methods and transformations developed in (i) to (iv) above to specific domains, particularly:

(a) Security, trust and privacy ontologies (Chang, Dillon & Hussain 2006),

(b) bioinformatics, such as in Sidu et al., (Sidhu, Dillon & al. 2005a, 2005b, 2006),

(c) an ontology based, software agent-driven multi-site software engineering (Wongthamtham, Chang & Dillon 2004), and

(d) ontology approach in Human Disease Studies (Hadzic & Chang 2004).

It should be noted that, in regards to points (i) and (ii) above, some initial research results and findings has been published in (Wouters 2006; Wouters et al. 2006). In this chapter, we focus on (i) – (iii) and (v) only.

9.5 Case Study Example

As in Chapter 6, we use the e-Sol case study to elaborate on our concepts presented in this chapter. Here, the case study is considered in the context of supporting an ontology-driven e-Sol. This is described in detail below.

In e-Sol, due to the nature of e-Sol and its business process, data semantics have to be in different formats (ontology bases and vocabularies) to support multiple systems, customers, warehouses and logistics providers. Also, data has to be duplicated at various points in time, in multiple databases to support collaborative business needs. In addition, since new customers and service providers join the e-Sol (or leave), the data semantics, description and formats have to be dynamic and should be efficiently duplicated without loss of semantics. This presents an opportunity to investigate how to integrate and utilize various customers' and collaborative partners' ontology bases for mutual benefit in the SW paradigm. The Figs. 9.1 – 9.2 illustrate some simplified ontology concepts developed for the e-Sol.

It should be noted here that in this chapter, to model conceptual views, we use our own notional semantics bases OMG's UML (OMG-UML™ 2003a) (for representing our concepts). This notation is similar to the one used in (Wouters 2006; Wouters et al. 2004b; Wouters et al. 2006). We use UML in this chapter as a conceptual modelling notation rather than as an ontology modelling notation, to demonstrate our concepts and applications and our intention is neither to emphasis nor promote the notation used as the only modeling notation for representing ontologies. A discussion of the modelling notation is outside the scope of this chapter and the thesis. However, there exist many other ontology modelling notations and paradigms, such as, UML Profiles (Gaševic, Djuric & Devedzic 2005; Dragan Gaševic et al. 2004a, 2004b), Object Role Modelling (ORM) (Jarrar, Demey & Meersman 2003; Spyns, Meersman & Mustafa 2002) , XSemantic nets (Wouters 2006), OWL (W3C-

OWL 2004) to name a few. A detailed discussion of ontology modelling notation can be found in (Wouters 2006).

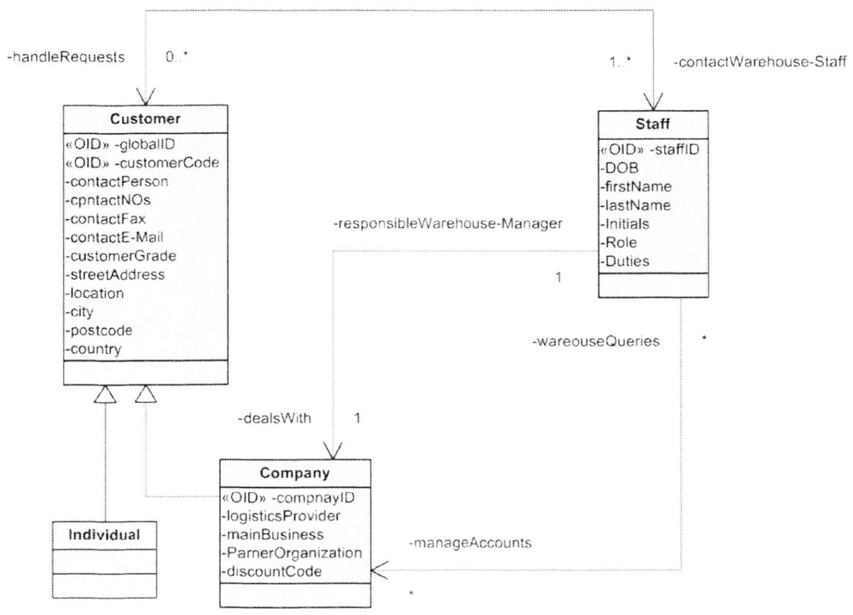


Figure 9.1: A simple ontology in the e-Sol (UML)

It should be noted that, since our main focus here is not modelling ontologies or modelling notations, the examples and the figures given for the e-Sol in this chapter are for illustrative purposes only and do not provide the complete ontology model of the system. Also, our use of OMG's UML (OMG-UML™ 2003a) is only for representational convenience, since its notion and semantics are understood by the wider community. In the related work, though not considered here, is the modelling of ontologies using XSemantics nets and the initial findings are published in (R.Rajugan, Chang, Feng et al. 2005).

9.6 Views for Semantic Web

The above issues provide a strong motivation to investigate *views* as a mechanism to extract and present sub-ontologies. In database systems, views (Elmasri & Navathe 2004) have proven to do similar tasks applicable to data warehousing and OLAP (Elmasri & Navathe 2004; Gopalkrishnan, Li & Karlapalem 1999; Mohania, Karlapalem & Kambayashi 1999; Nassis et al. 2005), where vast amount of historical data are cleaned, classified and loaded for analyzing and predictions. Here, views improve the quality of the sub-domain extracted and/or presented and the efficiency of processing queries.

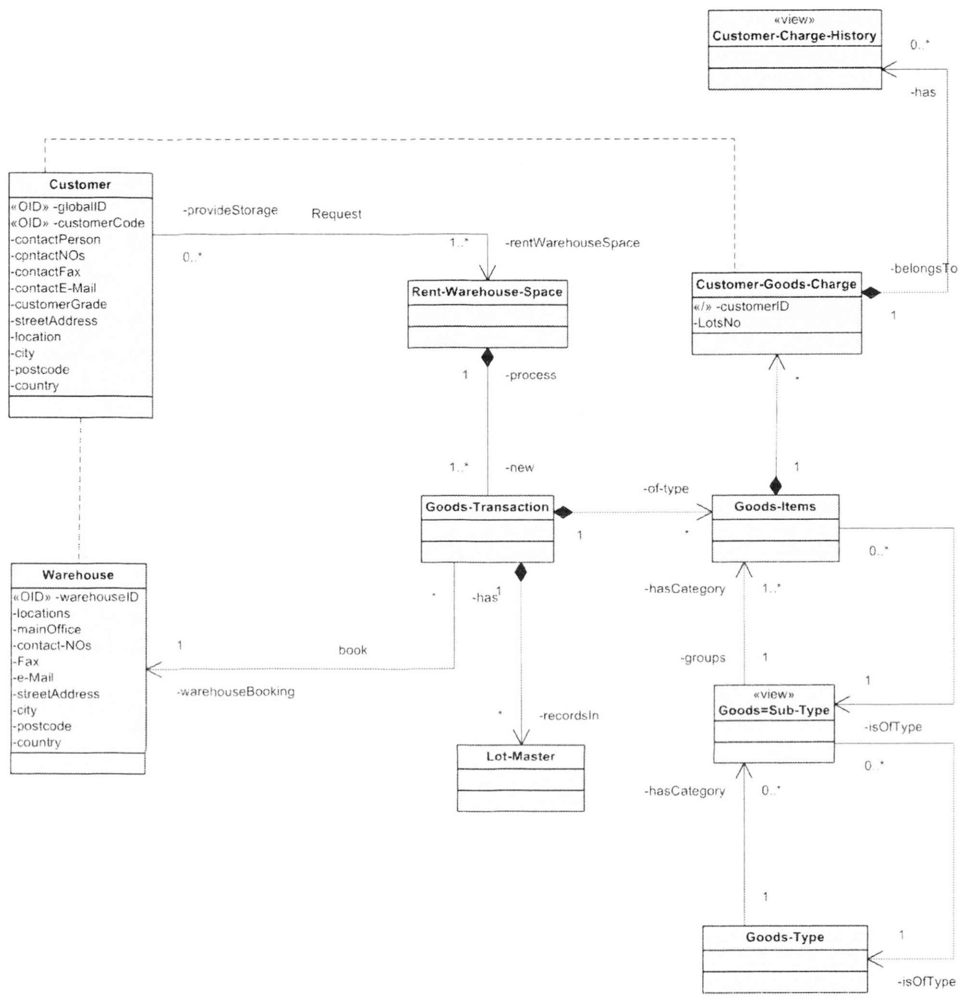


Figure 9.2: A simple ontology example in e-Sol

In our SW research, using our experience and knowledge gained from the LVM and the *view-driven* XDW model, we proposed to investigate a generic view model for SW, ontology views. In working with views for XML and XDW, we found that it is imperative to maintain the quality of the view (data) generated and without loss of semantics. This is one of the core requirements for ontology views as the ontology view (or the sub-ontology extracted from the main ontology base(s)) is complete, cohesive and consistent and satisfies the user requirement (Wouters et al. 2004b). Also, in direct contrast to data (or XML) views, ontology views should be useable as a standalone base ontology for the user and/or application by itself, independent of the parent/main ontology base. Also, unlike database views, given an ontology view, users and applications should be able to further customize it locally to suit his/her within the scope of the ontology standards and annotations.

Based on our work with XML views (namely semi-structured data domain) and some initial work done with views models for SW (R.Rajugan, Chang, Dillon et al. 2005; Wouters, R.Rajugan et al. 2005), we identified some key ***characteristics*** that ought to be present in a proposed *ontology views*. They are:

(i) Ontology view definitions should be based on strong (generic) theoretical foundations than are bound to a query and/or language specific syntax. This is a challenging task, as unlike data views, theoretical foundations for SW and ontologies are not yet strong enough to accommodate view definition and specifications. Though many research directions are focused on this area, adopting a generic theoretical framework work for ontology view definition is still an open challenge.

(ii) View specification and definition should be done (preferably) visually (e.g. such as view definitions using Query-by-Example frameworks developed for relational database models), at a higher level of abstraction due to the complexities involved in defining ontologies a lower level construct. Again, visualization of ontology bases are still in their infancy due to the complexities involved of doing such a computationally demanding task. This is mainly due to the scale (or size) and the complex inner structures (and the structural relationships) that are present in a typical ontology base.

(iii) Ontology views should provide facilities to be modelled in top-down (and preferably bottom-up as well) to conceptualize the sub-models and concepts. Providing such facilities enables the ontology engineer to design views with rich semantics and constraints. Some work has been done in this direction to model ontology bases (and in some cases views) using proven software engineering modeling techniques, such as Object-Oriented (OO) techniques using notations such as UML (Dragan Gaševic et al. 2004b; Wouters et al. 2004b), ORM (Jarrar, Demey & Meersman 2003; Spyns, Meersman & Mustafa 2002) or semantic nets (R.Rajugan, Chang, Feng et al. 2005).

However, high level modelling, design and querying techniques still prove to be a challenging task for SW paradigm as, ontology definitions and querying are more complex in nature than traditional modeling notations can model/represent.

(iv) Since SW models, namely ontologies, are defined and queried at a higher level of abstraction than at the data or instance level (Volz, Oberle & Studer 2003; Wouters et al. 2004a), the ontology view model should be specified using an abstract notation that is easily transferable to any language or platform-specific syntax.

(v) Since a given ontology view definition may be deployed in a variety of execution platforms, the view specification and definition must be portable. Also, an ontology view specification should be generic enough to be adaptable to new and emerging SW standards and annotations as the SW paradigm is still evolving with new languages, tools and platforms.

(vi) The ontology view derivation process should be optimized as future query execution schemes against the ontology view should provide optimal performance.

(vii) Given a main ontology which may contain distributed, inter-connected domains, ontology views should be able to distinguish between common concepts that may occur and do present in one or more inter-connected sub-domains. This is one of the major challenges faced in sub-ontology extraction.

(viii) Since an ontology view may substitute itself as a base for users and/or applications, it should provide facilities to be materialized (Wouters et al. 2004a) and extraction of further views (or sub sub-ontologies).

(ix) Given a *specific* ontology view, it should be able to be *commit* to the base (and the generic) ontology and reflect schematic changes as they occur.

Given the scope of this thesis, in the context of views for the Semantic Web, this chapter explores only the features (i) – (iv), as they are analogous to the views in the LVM. These are discussed in the following sections. The rest is explored as future research directions in Chapter 11.

9.6.1 Ontology Views

Views can be considered as special kind of transformation, namely the 3 E's (Wouters 2006): Extraction, Elaboration and Extension. Thus, an ontology view may be considered as a generic partitioning of any transformation into the 3 Es. That is:

- Extraction can informally be defined as taking a part of the original, so without any modifications.
- Elaboration is providing such an extracted part with additional levels of detail.
- Extension can be considered as the addition of completely new elements (i.e. they were not present in any shape or form in the original).

In the LVM model, since the view specification and the definitions are done at the highest level of abstraction (i.e. the conceptual level), support for the three Es are easily achievable task, as in the LVM, the view definitions are transferable (using transformations) across the required level of abstraction (i.e. conceptual, logical or document levels), without loss of semantics. However, to keep in line with the LVM semantics, we do not consider view elaborations or extensions.

Analogous to the LVM for XML, the LVM model for SW (Fig. 9.3), namely the ontology views, has three levels of abstraction. However, in the case of LVM for XML, there exists a clear distinction between three levels of abstraction, namely:

(a) conceptual,

(b) logical (or schematic), and

(c) document (or instance).

This is because, in a traditional data engineering approach, there exists a need to distinguish these levels of abstractions (Date & Darwen 1998; Dillon & Tan 1993;

Elmasri & Navathe 2004). But, in the case of ontology engineering (Gâomez-Pâerez, Fernâandez-Lâopez & Corcho 2003; Spyns, Meersman & Mustafa 2002), though there exists a clear distinction between conceptual and logical (or schema) models, the separation between the logical (or schema) level and the document (or instance) level tends to overlap due to the nature of ontologies, where concepts, relationships and values may be present in mixed sorts, such as schemas and values.

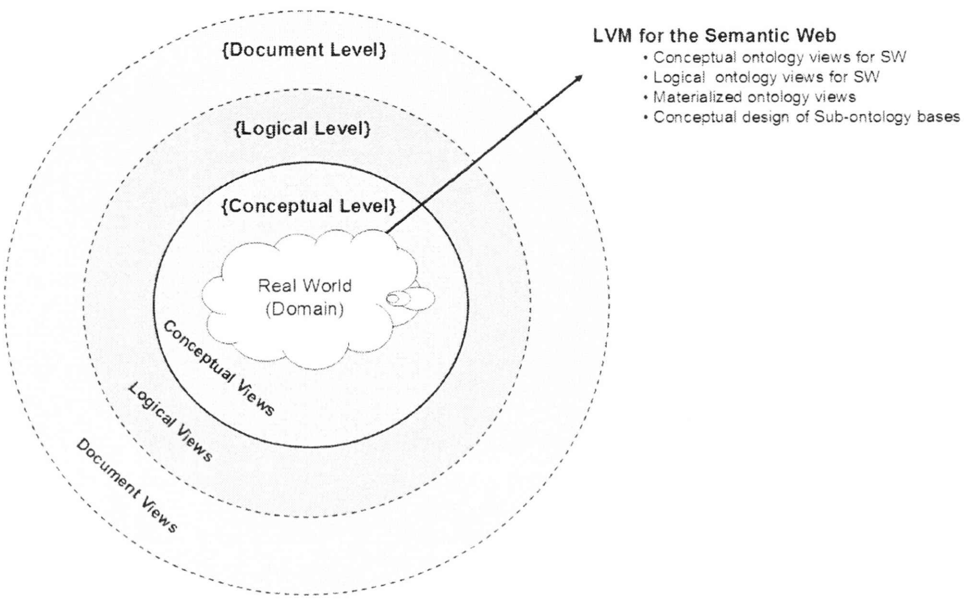

{Document Level}
{Logical Level}
{Conceptual Level}
Real World (Domain)
Conceptual Views
Logical Views
Document Views
LVM for the Semantic Web
• Conceptual ontology views for SW
• Logical ontology views for SW
• Materialized ontology views
• Conceptual design of Sub-ontology bases

Figure 9.3: LVM for Semantic Web (ontology views)

Therefore, here, in the LVM for SW (i.e. ontology views), as shown in Fig. 9.3, we provide a clear distinction between conceptual and logical views, but depending on the domain and/or application, we allow an overlap between the logical views and document views (thus it is shown as a dashed line in Fig. 9.3). This is one of the main differences between the LVM for XML and the LVM for SW.

To the best of our knowledge, other than our work, there exist no research directions that explore the LVM approach for the SW paradigm. Also, analogous to the LVM for XML, the notion of ontology views have explicit constraints and an extended set of expressive conceptual operators to support Ontology Extraction Methodology (OEM) (Wouters et al. 2002; Wouters et al. 2004a, 2004b), as presented later in section 9.7.

9.6.2 Ontology View Semantics & Definitions

In this section, we present some of the formal semantics associated with ontologies and ontology views that are synonymous with the views in the LVM. Since the proposed model is based on the LVM and the associated concepts, some of the concepts and definitions are extensions and/or elaboration of the concepts and definitions presented in Chapter 5.

There exist a few definitions that are specific for ontology. However, in this research we adopt, elaborate and extend the definitions provided by Wouters et al. in (Gruber 1993; Wouters 2006), since this is one of the very few early works that provides some form of conceptual abstraction in modelling and defining ontology bases and ontology views. It should also be noted that the discussion provided here are about formal semantics and extend some of the LVM semantics and properties to SW). Our intentions are neither to propose new definitions for existing ontology standards nor to propose formal semantics for existing standards.

First we define the notion of a general ontology, given in (Gruber 1993):

Definition 9.1. Ontology is an explicit specification of a conceptualization that corresponds to the Universe of Discourse (UoD) of the given domain.

In this research, an agent is an entity such that:

Postulate 9.1. An agent is a *participating* entity such as users, agents and software modules or applications in a Semantic Web environment.

As stated in Chapter 5, a *context* refers to the portion of the domain that is of interest. Thus, in the case of ontologies, analogous to the definition of in the LVM, we elaborate and state a context as;

Postulate 9.2. The term ***ontological context*** $o\zeta_{ontext}$ refers to a portion of the conceptual specification of a given domain (i.e. UoD) that is of interest to one or more agents.

From equation 5.5, we can show that:

$$o\zeta_{ontext} \in UoD \qquad (9.1)$$

Thus, we define the ontological context as:

Definition 9.2. An *ontological context* $o\zeta_{ontext}$ is a 4-ary tuple of $o\zeta_{name}, o\zeta_{obj}, o\zeta_{rel}$ and $o\zeta_{onstraint}$, where $o\zeta_{name}$ is the name of the context $o\zeta_{ontext}$, $o\zeta_{obj}$ is a set of objects in $o\zeta_{ontext}$, $o\zeta_{rel}$ is a set of object relationships in $o\zeta_{ontext}$, and $o\zeta_{onstraint}$ is a set of constraints associated with $o\zeta_{obj}$ and $o\zeta_{rel}$ in $o\zeta_{ontext}$.

$$o\zeta_{ontext} = (o\zeta_{name}, o\zeta_{obj}, o\zeta_{rel}, o\zeta_{constraints}) \qquad (9.2)$$

Here, in definition 9.2 above, we denote the ontology terminology defined in (Wouters 2006), such that,

(i) The set of objects in ontological context $o\zeta_{ontext}$ represents the notion of a set of concepts, as defined (Wouters 2006).

(ii) The set of objects in ontological context $o\zeta_{ontext}$ are finite; that is, the number of concepts in an ontological context is finite.

(iii) The set of objects in ontological context $o\zeta_{ontext}$ are a non-empty set.

(iv) The properties (or elements) of $o\zeta_{obj}$ corresponds to the set of all possible attributes $o\zeta_{attribute}$ (Wouters 2006) for the given concept

(v) The set of properties for a given context is always finite.

(vi) As in the case with the LVM, the $o\zeta_{rel}$ corresponds to the set of all possible *binary* relationships between one or more concepts and/or their attributes in a given ontological context $o\zeta_{ontext}$.

(vii) The $o\zeta_{onstraint}$ corresponds to a set of all possible constraints associated with the concepts, attributes and relationships for a given ontological context $o\zeta_{ontext}$.

However, in defining the terms above, here, we make some implicit assumptions about the restrictions that are placed in defining an ontology definition, similar to the restrictions applied (Wouters 2006), in the case of the Internal Ontology Conceptualization (IOC) as defined in (Wouters 2006).

Let we denote $\Re$ as a restriction that is applied on a basic ontology definition.

(i) Attribute Connection Restriction ($\Re_{ACR}$): Let $o\zeta_{attribute}$ denote an attribute set that belongs to a given *ontological context* $o\zeta_{ontext}$. An ACR implies that, given an attribute set, it always belongs to one concepts. That is, $o\zeta_{attribute}$ always belongs to a member of $o\zeta_{obj}$.

$$\Re_{ACR} \Rightarrow (\forall o\zeta_a) | (o\zeta_a \in o\zeta_{attribute} \wedge o\zeta_a \subset o\zeta_{obj}) \qquad \textbf{(9.3)}$$

(ii) Ontology Graph Restriction ($\Re_{OGR}$): As defined in (Wouters 2006), a graph of an *ontological context* $o\zeta_{ontext}$ does not contain *islands*.

(iii) Internal Mapping Restriction ($\Re_{IMR}$): In Wouters et al., the IOC ontology definition has an explicit mapping, which maps the concepts onto attributes. Here, in our work, we assume that the mapping is an internal restriction of the concepts that provides such mappings.

Thus, in this research the restrictions that apply to the ontological context include all of the above, written as:

$$\Re = \Re_{ACR} \cup \Re_{OGR} \cup \Re_{IMR} \tag{9.4}$$

Definition 9.3. An *ontological context* $o\zeta^{X}_{ontext}$ is said to be a valid ontology if and only if, it is a context with the restrictions $\Re$ are applied, such that:

$$o\zeta^{X}_{ontext} = (\forall o\zeta^{X}_{ontext}) \mid o\zeta^{X}_{ontext} \in o\zeta_{ontext} \wedge \Re(o\zeta^{X}_{ontext}) \tag{9.5}$$

Example 9.1: For example, some ontological contexts are given in Figs. 9.1 and 9.2. In Fig. 9.1, the set of concepts (represented using UML class) include $o\zeta^{X}_{ontext} = \{Customer, Staff, Company, Individual\}$, with $o\zeta_{name} = people$, and the attribute sets includes all the attributes of all the corresponding classes.

Let $o^{X}_{ntology}$ denotes an ontology.

Definition 9.4. An ontology $o^{X}_{ntology}$ is said be a sub-ontology of a valid ontological context $o\zeta^{X}_{ontext}$, if and only if:

$$o^{X}_{ntology} = (\forall o^{X}_{ntology}) \mid o^{X}_{ntology} \in o\zeta^{X}_{ontex} \wedge o^{X}_{ntology} \subset o\zeta^{X}_{ontex} \qquad \textbf{(9.6)}$$

From equation 9.4, it can be shown that, for $o^{X}_{ntology}$ to be a valid sub-ontology, it implies that:

$$o^{X}_{ntology} \Rightarrow o^{X}_{ntology} \in o\zeta_{ontext} \wedge \Re(o^{X}_{ntology}) \qquad \textbf{(9.7)}$$

We denote the sub-ontology as $\subset\subset$, thus:

$$o^{X}_{ntology} \subset\subset o\zeta^{X}_{ontex} \qquad \textbf{(9.8)}$$

Example 9.2: A valid sub-ontology $o^{X}_{ntology}$ of the ontological context $o\zeta^{X}_{ontext}$ is shown in Fig. 9.4, where, $o^{X}_{ntology} = \{Customer, Company, Individual\}$.

As defined in Chapter 5, analogous to the conceptual view definition in LVM, we define a *conceptual ontology view* as:

Postulate 9.3. A *conceptual ontology view* refers to a certain portion or sub-set of a given ontological context that is of interest to participating agents in the SW environment, at a given point in time.

Example 9.3: In Fig. 9.4, at a given point in time, an agent may only want to look at "people" who are customers in the e-Sol.

Example 9.4: In Fig. 9.5, at a given point in time, "Warehouse-Booking" and "Goods-Item-Classification" are valid conceptual ontology views.

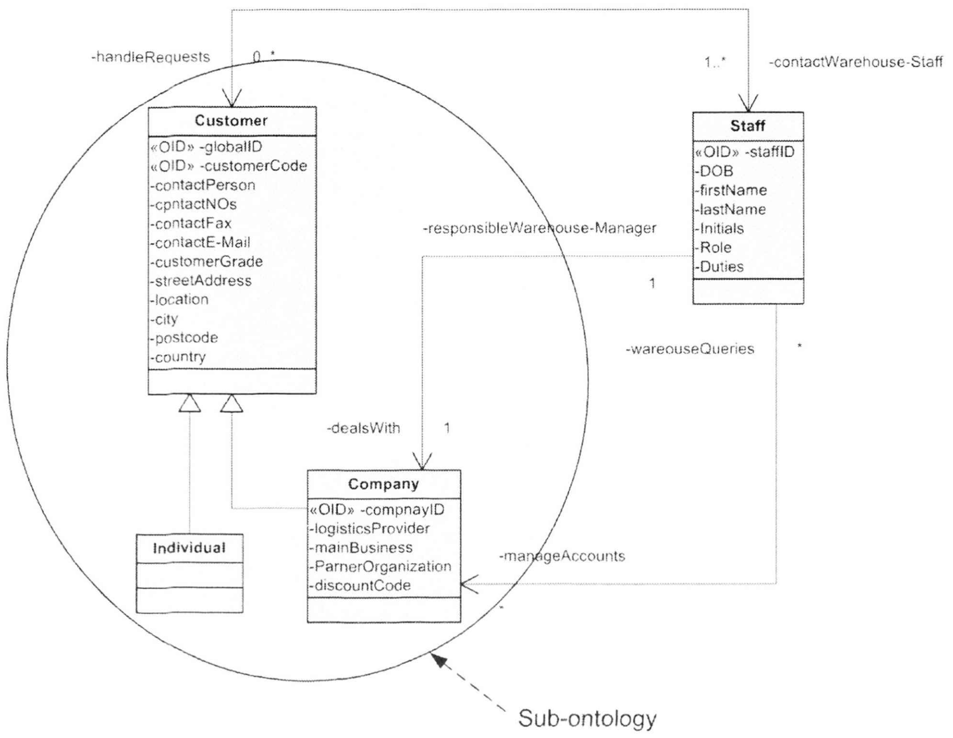


Figure 9.4: Ontology conceptual view / sub-ontology example in e-Sol

Also, from definition 5.2 (Chapter 5), we can derive that:

Definition 9.5. A *conceptual ontology view* oV_{iew}^{c} is a 4-ary tuple of $oV_{name}^{c}, oV_{obj}^{c}, oV_{rel}^{c}$ and $oV_{constraints}^{c}$, where oV_{name}^{c} is the name of the conceptual ontology view oV_{iew}^{c}, oV_{obj}^{c} is a set of objects in oV_{iew}^{c}, oV_{rel}^{c} is a set of object relationships in oV_{iew}^{c}, and $oV_{constraints}^{c}$ is a set of constraints associated with oV_{obj}^{c} and oV_{rel}^{c} in oV_{iew}^{c}.

$$oV_{iew}^{c} = (oV_{name}^{c}, oV_{obj}^{c}, oV_{rel}^{c}, oV_{constraints}^{c}) \qquad \textbf{(9.9)}$$

Let λ be a set of conceptual operators, as defined in Chapter 5.

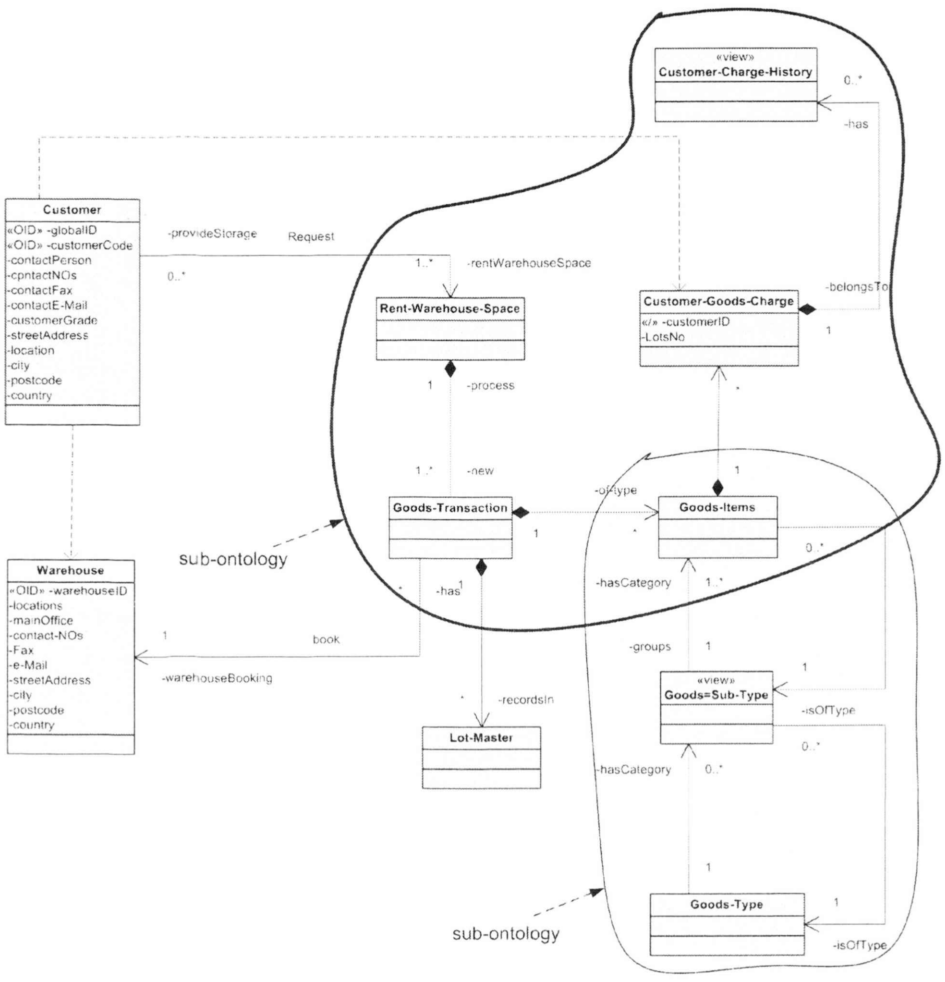


Figure 9.5: Ontology conceptual view / sub-ontology examples in e-Sol

Definition 9.6. A *conceptual ontology view* oV^{c}_{iew} is called a *valid* conceptual ontology view of the ontological context $o\zeta^{X}_{ontext}$, if and only if the following conditions (i), (ii), (iii) and (iv) are satisfied:

(i) A valid sub-ontology such that, $oV^{c}_{iew} \subset\subset o\zeta_{ontext}$

(ii) For any object $\forall o^{v}_{obj} \in oV^{c}_{iew}$, there exist objects $\exists o^{c1}_{obj}, o^{c2}_{obj},..o^{ci}_{obj},..o^{cn}_{obj} \in o\zeta_{obj}$, such that, $o^{v}_{obj} = \lambda_1\, \lambda_2,..,\lambda_m\, (o^{c1}_{obj}, o^{c2}_{obj},..o^{ci}_{obj},..o^{cn}_{obj})$ where

$\lambda_1\,\lambda_2,...,\lambda_m \in \bar{\lambda}$. That is, o^{v}_{obj} is a newly derived object from existing objects $o^{c1}_{obj}, o^{c2}_{obj},.......o^{cn}_{obj}$ in the ontological context via a series of conceptual operators $\lambda_1\,\lambda_2,...,\lambda_m$.

(iii) For any constraint $\forall c^{v}_{onstraint} \in oV^{c}_{constraints}$, there exists a constraint $\exists c^{v'}_{onstraint} \in o\zeta_{constraint}$ or a new constraint $c^{v''}_{onstraint}$ constraints associated with V^{c}_{obj} and V^{c}_{rel}.

(iv) For any hierarchical relationship $\forall rel_h \in oV^{c}_{rel}$, there *does not exist* a relationship between one or more V^{c}_{obj} and ζ_{obj}.

(v) For any binary association relationship/dependency relationships $\forall rel_a \in oV^{c}_{rel}$, there *may exist* a relationship between one or more oV^{c}_{obj} and $o\zeta_{obj}$.

Example 9.5: In Fig. 9.1, at a given point in time, an agent may want to look only at "`people`" who are customers in the e-Sol. Thus we can have a conceptual ontology view, such that (as shown in Fig. 9.4, using UML.);

$$\begin{aligned} &oV^{c}_{name} = cutomers \text{ with} \\ &oV^{c}_{obj} = \{Customer, Company, Individual\} \end{aligned} \qquad \textbf{(9.10)}$$

It should be noted here that we did not present a detailed discussion and logical and document level ontology view semantics as it largely dependent on the underlying metadata description and/or representation languages such as RDF, RDF-S, OWL, etc. Though we intend to address this issue in the near future, at present we use other semantics and transformation methodologies proposed to carry out the task of mapping conceptual ontology views to logical and/or document views. One such transformation is discussed in the next section and the transformation considered includes, UML-to-OWL (Dragan Gaševic et al. 2004b; D. Gaševic et al. 2004) transformation, ORM-to-OWL (Jarrar, Demey & Meersman 2003), etc.

9.7 Transformation Ontology View to the Materialized Ontology View Extractor (MOVE) System

The MOVE system (Wouters 2006; Wouters et al. 2004a) is a deployment architecture that allows meaningful and automated extraction of sub-ontologies under the Ontology Extraction Methodology (OEM) (Wouters 2006; Wouters et al. 2006). The resulting MOVE system views are high-quality, materialized and *optimized* sub-ontologies. One of the main benefits of the MOVE system is that it can be used by domain experts who are not familiar with complex software and/or database related concepts and jargons. Also, it does not need ontology experts to improve the quality of the resulting ontology views as it is guaranteed to provide quality views.

The MOVE system, looks at sub-ontologies as a case in point, and provides:

(i) Sub-ontology extraction;

(ii) Optimization schemes

  (a) Requirement Consistency Optimization Scheme (RCOS)

  (b) Well-Formedness Optimization Scheme (WFOS)

  (c) Semantic Completeness Optimization Scheme (SCOS)

  (d) Total Simplicity Optimization Scheme (TSOS)

  (e) Total Versatility Optimization Scheme (TVOS)

  (f) Medium Simplicity Optimization Scheme (MSOS)

(iii) Re-labelling

In the MOVE, views are referred to as "*strict*" (Wouters et al. 2006) as they are focused on extraction only and do not provide for elaboration. This is because, as opposed to LVM views for XML, all the conceptual ontology views that can be constructed using one or more conceptual operators would not constitute a valid ontology view (Wouters et al. 2006). However, the result of the OEM extraction process is always a valid conceptual ontology view that corresponds to a strict ontology view in the MOVE system.

Definition 9.7. A *strict* ontology view in the LVM is a materialized sub-ontology in the MOVE system that is derived from an ontology (called the base ontology). The derivation can consist of any (combination) of the following operations; synonymous rename, selection and compression (Wouters 2006; Wouters et al. 2006).

In order to transform our ontology views to the MOVE system using the OEM approach, here, the conceptual operators described in Chapter 5 are mapped to synonymous MOVE system (conceptual) operators. For example, for the conceptual operators in the LVM model:

- The MOVE (synonymous) rename operator, which is a restricted version of the LVM rename unary operator: a synonymous rename operator of an element indicates that the same element is still present in the ontology view, but in a (syntactically) alternative form. Semantically, however, the same or less information is present.
- The MOVE (synonymous) selection operator, which is a selection operator in the LVM: a selection of an element indicates that that element is still present in the ontology view, in unaltered form. This is similar to the selection operation executed over a ontological context, in the LVM
- The MOVE compression operator: A compression of elements indicates that those elements are replaced by a single element in the ontology view. The element itself can be a new element, but it will not provide additional semantic information (compared to the base ontology). The compression operator is a complex operator, constituted of the concatenation of multiple unary operations applied in sequence.

However, it should be noted here that the application of LVM binary operators is not considered in the MOVE system.

9.8 Conclusion

In this chapter, we presented an intuitive research direction to extend the LVM concepts to the Semantic Web. First, we presented the Semantic Web and related terminologies, including some discussion of the difference between the traditional data centred domain concepts and the ontological domain concepts. Then we described our proposed adoption of the LVM for the Semantic Web, its properties and semantics. This includes some formal semantics and definitions associated with the proposed LVM for the Semantic Web, with use of the case study examples. Later, we describe one solution to deploying the proposed views in the MOVE system, by providing a transformation to map LVM conceptual operators to MOVE system operators.

In the following chapter, we will continue to describe another case study and applications of the LVM in the context of website and web portal design (Chapter 10). Chapter 11 provides a recapitulation of the main aspects of this thesis with a discussion of future research directions including addressing some outstanding issues described in this chapter.

References

Aberer, K, Catarci, T, Cudré-Mauroux, P, Dillon, TS, Grimm, S, Hacid, M-S, Illarramendi, A, Jarrar, M, Kashyap, V, Mecella, M, Mena, E, Neuhold, EJ, Ouksel, AM, Risse, T, Scannapieco, M, Saltor, F, De Santis, L, Spaccapietra, S, Staab, S, Studer, R & De Troyer, O 2004, 'Emergent Semantics Systems', *First International IFIP Conference on Semantics of a Networked World (ICSNW 2004) - Revised Selected Papers*, vol. 3226, ed. CAG Mokrane Bouzeghoub, Vipul Kashyap, Stefano Spaccapietra, Springer, Paris, France, pp. 14-43.

Berners-Lee, T 1999, 'Weaving the web', Harper, San Francisco.

Bhatt, M, Flahive, A, Wouters, C, Rahayu, W, Taniar, D & Dillon, TS 2004, 'A Distributed Approach to Sub-Ontology Extraction', *18th International Conference on Advanced Information Networking and Applications (AINA'04) - Volume 1*, vol. 1, IEEE Computer Society, Fukuoka, Japan, pp. 636-41.

Chang, E, Dillon, TS & Hussain, F 2005, *Trust and Reputation for Service Oriented Environments - Technologies for Building Business Intelligence and Consumer Confidence*, John Wiley & Sons, London, UK.

Chang, E, Dillon, TS & Hussain, FK 2006, *Trust and Reputation for Service Oriented Environments: Technologies for Building Business Intelligence and Consumer Confidence*, John Wiley and Sons, UK.

Cruz, I, Decker, S, Euzenat, J & McGuinness, D (eds) 2002, *The emerging semantic web : selected papers from the first Semantic Web Working Symposium*, IOS Press,Tokyo : Ohmsha.

Date, CJ & Darwen, H 1998, *Foundation for object/relational databases : the third manifesto : a detailed study of the impact of objects and type theory on the relational model of data including a comprehensive proposal for type inheritance*, Addison-Wesley, Reading, Mass.

Dillon, TS & Tan, PL 1993, *Object-Oriented Conceptual Modeling*, Prentice Hall, Australia.

Do, HH & Rahm, E 2004, 'Flexible integration of molecular-biological annotation data: The genmapper approach', *Proceedings of the 9th International Conference on Extending Database Technology (EDBT '04)*, Heraklion, Crete, Greece.

Elmasri, R & Navathe, S 2004, *Fundamentals of database systems*, 4th edn, Pearson/Addison Wesley, New York.

Gâomez-Pâerez, A, Fernâandez-Lâopez, M & Corcho, O 2003, *Ontological engineering : with examples from the areas of knowledge management, e-commerce and the semantic Web*, Advanced information and knowledge processing., Springer-Verlag, London ; New York.

Gaševic, D, Djuric, D & Devedzic, V 2005, 'Bridging MDA and OWL Ontologies', *Journal of Web Engineering*, vol. 4(2), no. 2, pp. 118-43

Gaševic, D, Djuric, D, Devedzic, V & Damjanovic, V 2004a, 'Approaching OWL and MDA Through Technological Spaces', *3rd Workshop in Software Model Engineering (WiSME 2004)*, Lisbon, Portugal.

Gaševic, D, Djuric, D, Devedzic, V & Damjanovic, V 2004b, 'Converting UML to OWL Ontologies', *Proceedings of the 13 th International World Wide Web Conference*, NY, USA, pp. 488-9.

Gaševic, D, Djuric, D, Devedžic, V & Damjanovic, V 2004, 'UML for Read-To-Use OWL Ontologies', *Proceedings of the IEEE International Conference Intelligent Systems*, Vrana, Bulgaria.

Gopalkrishnan, V, Li, Q & Karlapalem, K 1999, 'Star/Snow-flake Schema Driven Object-Relational Data Warehouse Design and Query Processing Strategies', *1st First International Conference on Data Warehousing and Knowledge Discovery (DaWaK '99)*, Springer, Florence Italy.

Gruber, TR 1993, 'A Translation Approach to Portable Ontologies', *Knowledge Acquisition*, vol. 5, no. 2, pp. 199-220

Hadzic, M & Chang, E 2004, 'Role of the Ontologies in the Context of Grid Computing and Application for the Human Disease Studies', *Semantics for Grid Databases, First International IFIP Conference on Semantics of a Networked World (ICSNW '04)*, vol. LNCS 3226, Springer Verlag, Paris, France, pp. 316-8.

Jarrar, M, Demey, J & Meersman, R 2003, 'On Using Conceptual Data Modeling for Ontology Engineering', *Journal on Data Semantics*, vol. LNCS 2800, pp. 185-207

Luk, RWP, Leong, HV, Dillon, TS, Chan, ATS, Croft, BW & Allan, J 2002, 'A survey in indexing and searching XML documents', *Journal of the American Society for Information Science and Technology*, vol. 53(6), no. 6, pp. 415-37

Mohania, MK, Karlapalem, K & Kambayashi, Y 1999, 'Data Warehouse Design and Maintenance through View Normalization', *10th International Conference on Database and Expert Systems Applications (DEXA '99)*, vol. 1677, Springer, Florence, Italy, pp. 747-50.

Nassis, V, R.Rajugan, Dillon, TS & Rahayu, W 2005, 'Conceptual and Systematic Design Approach for XML Document Warehouses', *International Journal of Data Warehousing and Mining*, vol. 1(3), no. 3, pp. 63-87

Noy, N & Musen, M 2002, 'Promptdiff: a fixed-point algorithm for comparing ontology versions', *Proceedings of the Eighteenth National Conference on Artificial Intelligence*, American Association for Artificial Intelligence, pp. 744-50.

OMG-UML™ 2003a, *UML 2.0 Final Adopted Specification (http://www.uml.org/#UML2.0)*, OMG, 2005.

OMG-UML™ 2003b, *Unified Modeling Language™ (UML) Version 1.5 Specification*, OMG.

R.Rajugan, Chang, E, Dillon, TS, Feng, L & Wouters, C 2005, 'Modeling Ontology Views: An Abstract View Model for Semantic Web', *1st International IFIP WG 12.5 Working Conference on Industrial Applications of Semantic Web (IASW '05)*, Springer IFIP Book Series, Jyvaskyla, Finland, pp. 227-46.

R.Rajugan, Chang, E, Feng, L & Dillon, TS 2005, 'Modeling Views for Semantic Web', *First IFIP WG 2.12 & WG 12.4 International Workshop on Web Semantics (SWWS '05), In conjunction with On The Move Federated Conferences (OTM '05)*, Springer-Verlag, Agia Napa, Cyprus, pp. 936-46.

Sidhu, AS, Dillon, TS & al., e 2005a, 'Ontological Foundation for Protein Data Models', *First IFIP WG 2.12 & WG 12.4 International Workshop on Web Semantics (SWWS '05), In conjunction with On The Move Federated Conferences (OTM '05)*, Springer-Verlag, Agia Napa, Cyprus.

Sidhu, AS, Dillon, TS & al., e 2005b, 'Protein Ontology: Vocabulary for Protein Data', *3rd IEEE International Conference on Information Technology and Applications (IEEE ICITA 2005)*, vol. 1, IEEE CS Press, Sydney, Australia, pp. 465-9.

Sidhu, AS, Dillon, TS & al., e 2006, 'Protein Ontology: Data Integration using Protein Ontology', in Z Ma & JY Chen (eds), *Database Modeling in Biology: Practices and Challenges*, Springer, New York, NY, p. to appear.

Spyns, P, Meersman, R & Mustafa, J 2002, 'Data Modeling Versus Ontology Engineering', *SIGMOD*, pp. 14-9.

Volz, R, Oberle, D & Studer, R 2003, 'Views for light-weight Web ontologies', *Proceedings of the ACM Symposium on Applied Computing (SAC '03)*, ACM Press New York, NY, USA, USA, pp. 1168 - 73.

W3C-OWL 2002, *OWL web ontology language 1.0 reference*, W3C.

W3C-OWL 2004, *Web Ontology Language (OWL), (http://www.w3.org/2004/OWL/)*, The World Wide Web Consortium (W3C), viewed March 2005 http://www.w3.org/2004/OWL/.

W3C-RDF 2004, *Resource Description Framework (RDF), (http://www.w3.org/RDF/)*, The World Wide Web Consortium (W3C), viewed March 2000.

W3C-SW 2005, *The Semantic Web (http://www.w3.org/2001/sw/)*, W3C.

W3C-XQuery 2004, *XQuery 1.0: An XML Query Language (http://www.w3.org/TR/xquery)*, The World Wide Web Consortium (W3C), viewed November 2003.

Wongthamtham, P, Chang, E & Dillon, TS 2004, 'Methodology for Multi-site Software Engineering Using Ontology', *Proceedings of the International Conference on Software Engineering Research and Practice (SERP '04)*, CSREA Press, pp. 477-82.

Wouters, C 2006, 'A Formalization and Application of Ontology Extraction', Doctor of Philosophy (Ph.D) thesis, La Trobe University, Melbourne, Australia, Melbourne.

Wouters, C, Dillon, TS, Rahayu, JW & Chang, E 2002, 'A Practical Walkthrough of the Ontology Derivation Rules', *Database and Expert Systems Applications : 13th International Conference (DEXA '02)*, Springer, Aix-en-Provence, France, pp. 259-68.

Wouters, C, Dillon, TS, Rahayu, JW & Chang, E 2005, 'Large scale ontology visualisation using ontology extraction', *International Journal of Web and Grid Services*, vol. 1(1), no. 1, pp. 113 - 35

Wouters, C, Dillon, TS, Rahayu, JW, Chang, E & Meersman, R 2004a, 'Ontologies on the MOVE', *9th International Conference on Database Systems for Advanced Applications (DASFAA '04)*, vol. 2973/2004, eds Y Lee, J Li, K-Y Whang & D Lee, Springer-Verlag GmbH, Jeju Island, Korea, pp. 812-23.

Wouters, C, Dillon, TS, Rahayu, JW, Chang, E & Meersman, R 2004b, 'A Practical Approach to the Derivation of a Materialized Ontology View', in D Taniar & W Rahayu (eds), *Web Information Systems*, Idea Group Publishing, USA.

Wouters, C, R.Rajugan, Dillon, TS & Rahayu, JW 2005, 'Ontology Extraction Using Views for Semantic Web', in D Taniar & W Rahayu (eds), *Web Semantics and Ontology*, Idea Group Publishing, USA.

Wouters, C, R.Rajugan, Dillon, TS & Rahayu, JW 2006, 'Ontology Extraction Using Views for Semantic Web', in D Taniar & W Rahayu (eds), *Web Semantics and Ontology*, Idea Group Publishing, USA, pp. 01 - 40.

Chapter 10

Case Study & Applications III: View-Driven Website & Web Portal Design

10.1 Introduction

In the previous two chapters, two applications of the Layered View Model (LVM) were presented. In this chapter, we present another and the final such case study application: a conceptual framework for modelling, designing and transforming website and web portals using the LVM framework. Here, we look at modelling web content (and web navigational aspects (Gardner 2006)) and representing them using views in the LVM, independent of presentation semantics. The focus of this chapter is to present web content modelling and representation issues. The modelling and representation of (user) web navigation and user access control is the subject of a separate PhD thesis (Gardner 2006). Here, we are primarily using views in the LVM to demonstrate the use of views to support MDA (OMG-MDA 2003) like framework work to develop websites and web portals definitions (and distribution of) for web content.

The LVM based conceptual framework to develop websites at varying levels of abstraction is an *architectural framework*, called eXtensible Web (or *x*Web in short). The conceptual framework that enable one to model and design (web) portals in the *x*Web framework is referred to as the eXtensible Portal (or *x*Portal in short), again using the views in the LVM. In contrast to the traditional website (and portal) modelling and design, the *view-driven* *x*Web (and *x*Portal) research is focused on;

(a) Modelling web content independent of presentation oriented web languages such as HTML (W3C-HTML 1997),

(b) Modelling web user interactions independent of web languages and applications and

(c) Representation (and dissemination) of combined web content and web navigation definitions in a web platform and web language independent of storage structure.

In the following sections, we describe this research in detail.

This chapter is organized as follows. Section 10.2 looks at web content and website development in general. In section 10.3, the motivation for this research (i.e. eXtensible web/portal) is presented followed by discussion on some of the related work and background information in section 10.4. In section 10.5, we describe a case study example that is used to illustrate our concepts in this chapter. Sections 10.6 – 10.7 provide a detailed discussion on *x*Web research, starting from general concepts (section 10.6) and formal semantics in section 10.7. Section 10.8 provides an illustrative practical walkthrough of the *x*Web/*x*Portal design steps and the methodology and steps. Section 10.9 concludes this chapter.

10.2 Web Engineering

In software engineering, many methodologies have been proposed to capture real-world problems into manageable segments, which can be communicated, modelled and developed into error-free maintainable software models and modules (Booch 1993; Dillon & Tan 1993). Similarly, in the case of web engineering, web content, web access (or usability) and specification should represent a meaningful unit of information with respect to the semantics of the domain in question (Conallen 1999, 2003).

Since the early days, web engineering and website development have evolved from coding simple HTML based static pages into complex a software engineering discipline. The traditional web engineering techniques, which were based around textual file based structures, provided only limited or no facilities for modelling higher-level design concepts that go beyond the granularity of file-based textual information (Abiteboul et al. 2002; Aragones & Hart-Davidson 2002; Gaedke, Segor & Gellersen 2000; Lei, Motta & Domingue 2002). But, today's websites do not just deliver static content, but also support (web) application driven data transactions to complex multimedia web contents (Abiteboul et al. 1999; Lei, Motta & Domingue 2004b; W3C-WS 2002) and web services (W3C-WS 2002). Thus, with complex web contents such as interactive and hypermedia-based web sites, designers go beyond traditional HTML files and turn towards middleware and scripting technologies, from Common Gateway Interface (CGI) scripts to advanced SOAP messages. However, while there has been rapid advancement in the web implementation and deployment technologies, lack of sufficient modelling and design techniques that can provide a comprehensive modelling and design methodology, similar to those available for the development of classical (non-web based) software solutions still proves to be a challenge in the context of web engineering. This is mostly due to some of the unique characteristics of the web such as (Gardner 2006; R.Rajugan et al. 2005a):

(i) the nature of web content and its semantics (e.g. unstructured and/or semi-structured data with mixed content with no organized standards). Also, the unstructured and heterogeneous web contents are described using mainly presentation oriented mark-ups or languages;

(ii) the (unknown) end-user base (for e.g. number of users, skill-level, user-training, etc) and novice (i.e. untested or trained) end-user skills;

(iii) the stateless user or transaction session due to the nature of presentation and distribution centred web (network) architecture (protocols, standards etc.);

(iv) the unknown end-user platform and configuration. That is it is almost impossible to develop and test a web solution that is acceptable to all.

Web content may range from static un/semi-structured textual data to binary multimedia streams and dynamic on-the-fly hypermedia contents. Another dimension to web content is that they are distributed and hosted by multiple, geographically distributed servers and databases. In contrast to database managed structured data (relational or Object-Relational or OO), the web content do not conform to traditional data models which assume a fixed data model/schema for a given set of data domain. Therefore, since the introduction of Internet and the World-Wide-Web, researchers, standard organizations and the Industries rallied around to adopt a web data language that is semantically rich and descriptive yet conforms to an open standard schema, which can support and describe all types of data and content on the web, including traditional structured data. With XML/Schema the web is one step closer to achieving this goal. Therefore, since the beginning of the dot.com boom, researchers have proposed various web engineering techniques and methods to make web engineering an achievable dream by addressing one or more of the points (i) to (iv) described above.

In the case of web user-interface engineering, which is an important and integral part of websites, due to the unique characteristics of web architecture, it requires a great degree of flexibility in the design of the Web User Interface (WUI) (Gardner 2006) compared with the user interface design of traditional software applications. This, in turn, has had some undesirable consequences. Thus, similar to the development of any software application, a systematic approach is required to support the design process of WUI which in turn supports the web users to complete their tasks efficiently and effectively. Since it is outside the scope of this thesis to discuss web user interface problems and models, we direct the interested reader to a related work to *x*Web/*x*Portal research, namely in (Gardner 2006), where this issue, the web user interface problem, and proposed solutions, are addressed in detail.

In addition, similar to the importance of web engineering methodologies, it is also important to have uniform web standards (such as protocols, languages such HTML, XML etc) that are capable of handling heterogeneous web content, but also support emerging new engineering processes. To address some of these issues, since the late '90s, many approaches were proposed and tried, namely:

(i) Proposal of new web content description and modelling languages, standards and architectures;

(ii) Objected-Oriented (OO) like approaches for website (and web content) modelling and management using markup languages and databases;

(iii) Database (mostly relational) and middleware powered web content hosting (e.g. Oracle™ WebDB and later Oracle Portal[1]); and

(iv) Combination (i) – (iii) above.

However, to the best of our knowledge, none of the approaches provides a comprehensive, yet generic solution that helps web engineers from the conceptual modelling to implementation stage. Many web engineering solutions (for a simple web site design to e-Commerce engineering) are focused mainly on building, designing and maintaining web pages (using various styles and techniques) to support user-requests (from displaying simple text information to B2B and B2C transactions) without considering all four combined aspects of web engineering, namely:

(i) a well defined design methodology (such as OO) using a standard modelling notation (such as UML) to capture and model

- (a) (web) domain requirements,
- (b) web user requirements,
- (c) web user interface requirements (user-interface engineering),
- (d) web data requirements (data engineering) and
- (e) platform/architecture requirements;

(ii) a generic architecture adaptable to various implementation techniques;

(iii) well defined generic data standard (such as XML) to describe and represent web data; and

(iv) a process to address post-development maintenance and expansion.

Though there exist some tools that can perform one or two of the above stated aspects in regard to web engineering, to our knowledge, no tool or technique provides

[1] http://www.oracle.com/technology/products/ias/portal/index.html

all four aspects. Today, it is commonly agreed that a web architecture for the deployment of a meaningful website is the 3-tier architecture, which consists of the presentation layer, application layer, and data layer (Gardner et al. 2004; R.Rajugan et al. 2005a).

10.3 Motivation

In recent years, many research directions are revisiting the issue of website engineering to investigate metadata semantics for web content that is independent of the presentation oriented web page mark-ups. Such momentum was initially created by the Semantic Web initiatives (W3C-SW 2005), where domain specific web content are described using metadata languages such as XML, OWL (W3C-OWL 2004), RDF (W3C-RDF 2004), etc. Though useful, it is a complex task to provide a presentation independent, yet generic, semantically rich web content description for small scale websites (and web portals). Also challenging are the web useability concerns (Gardner 2006) that is core to the success of the web (and the Semantic Web). Here, in this chapter, we try to address part of this problem, the web metadata description and (generic) storage structure for website content.

It should be noted that the research presented in this chapter is unique in nature as it looks at modelling and representing web content using web usability modelling. Therefore, in the next section (section 10.4), to make it easier for the reader to follow some of the concepts presented in this chapter, we provide an extensive discussion that includes some of the usability concerns and user interface design related research, in the context of the modelling websites.

Thus, the core issues to this research in the context of website (and portal) design are:

(i) (web user) context selection (Gardner 2006)

(ii) (web user) task design (Gardner 2006)

(iii) web content (data) representation

Here, (i) and (ii) are addressed in detail as part of another thesis in (Gardner 2006). In (iii), we can address the use of LVM such that, the LVM view semantics are aligned with the semantics of the data repository which is used to populate the web pages. Thus, the LVM has a larger role in here.

The main motivation for the *x*Web (and *x*Portal) research includes:

(i) Web Content: The separation of website and web portal content from presentation oriented languages such as HTML, embedded scripts (e.g. JavaScript, server pages, etc.)

(ii) User Navigation: A top-down web user interaction analysis and modelling using proven user interface engineering to develop robust and user friendly web user interfaces (Elizabeth Chang 1996; Gardner, Chang & Dillon 2003a, 2003b) to support websites and web portals

(iii) Design methodology: A top-down website modelling and design approach using Object-Oriented (OO) conceptual modelling techniques

(iv) Generic model: A generic storage and description model to represent web content (in the form of website and web portal definitions)

(v) XML: Utilization of XML (and XML Schema) as the metadata and web content description language

(vi) Views in LVM: Utilization of LVM to support (i), (iii), (iv) and (v) above to develop a design methodology that is analogous to the MDA approach.

10.4 Background

Here we look at some work done in web design approaches and, in particular, website and web portal design, as our work incorporates both. There exist many works and tools for dealing with one or more aspects of general web engineering principles in works such as (Ceri, Fraternali & Bongio 2000; Elizabeth Chang & Dillon 1994; Conallen 2003; Gaedke, Segor & Gellersen 2000; Troyer & Leune 1998), but none of these address the whole spectrum of website development issues. In (Gu, Henderson-Sellers & Lowe 2002), the authors have argued that there are two aspects of technical architecture that a web modelling language must possess in order for it to be used effectively on the development of web systems, namely information architecture and functional architecture. Over the years, several techniques have been introduced in the literature for the modelling and designing of web-based systems. There is a heavy concentration in the earlier methods on:

(i) hypertext oriented (Garzotto, Paolini & Schwabe 1993; Schwabe, Rossi & Barbosa 1996);

(ii) data centric or data driven (Ceri, Fraternali & Bongio 2000).

It should be noted that these systems, hypertext, or data centered approaches need to be differentiated with the user-centred approach (Elizabeth Chang & Dillon 1999; Gardner, Chang & Dillon 2003a; Troyer & Leune 1998). Some of the more recent methods have its base on object-oriented paradigm (Conallen 2003). These models were found to not pay sufficient attention to users, who are central in web systems.

In regards to web portals, there exist many works that deal with the possibility of applications of portals in different areas (or specialized domains, such as in education, e-Sciences, etc.) of use (Gant & Gant 2002), and the classification and discussion of different types of portals (Reynolds, Shabajee & Cayzer 2004; Tatnall 2004). Only a few works looked into the issue of the actual design and development of web portals. One of the most interesting work includes (Bellas, Fernández & Muiño 2004), where the authors looks at the development of portal from a software engineering perspective. In (Aragones & Hart-Davidson 2002), usability issues are taken into account and the importance of evaluating these on customizable portals is also discussed.

However, most of these works do not provide a comprehensive design and technological solution for addressing both web data and web user interface design issues under one design methodology. We argue that such a combined design methodology is a must for any web system development such as re-usable websites and portals. In the related literature, there is a lack of consideration given to the idea that the implementation of a web user interface (WUI) is quite different from that of a traditional software system, as traditional software GUI is mainly constructed through the use of GUI widgets (Gardner, Chang & Dillon 2003a). Also, the kinds of devices that are used for the display of WUI are very diverse, such as PDAs, mobile phone, etc.

An interesting new development in web engineering is the utilization of ontologies. Works such as (Chan, Dillon & Siu 2002; Do & Rahm 2004; Gruber 1993; Lei, Motta & Domingue 2002, 2004a, 2004b) looked at the problem of web engineering (and web portal) from an ontology engineering point of view and some works have successful implementations. Though this is a good problem to solve for a domain-specific website engineering (such as the GONG project, http://gong.man.ac.uk/), it is not practical and economical for using ontologies to model and design generic and/or domain independent websites, because:

(i) Ontology-driven web engineering is a relatively new direction, and with the existing skills base (designers, modelers, technology, etc.) and the technologies, it is difficult to achieve MDA like generic website modelling and design methodologies.

(ii) It is a time consuming and complex task to develop and maintain a meaningful ontology base to support web content (for each domain where we need to develop a web site).

(iii) Also, combing generic website semantics into such an ontology base will increase the size and complexity of the ontology base beyond *practical* use, as complex query processing and sub-ontology extraction algorithms are required to construct the website pages.

(iv) Designing an ontology base for describing and representing web user interaction/profile is a challenging task and it is still and ongoing research problem (Gant & Gant 2002; Gardner, Chang & Dillon 2003a; Lei, Motta & Domingue 2002). Thus, the combining of web content description and web user interaction modelling is a complex and challenging task

In general, many of the more recent web design methods attempt to address navigational design partly in a way that is within the proposed process. However, the navigational model is often a by-product of the underlining domain model, which does not always provide the user view required as the user would like to perceive the information (Gardner, Chang & Dillon 2003a, 2003b). Rather, it only maps this data model to a suitable representation of the data for storage that is efficient for system manipulation, directly onto the presentation layer. It can be observed that there is the assumption that all data sources come from one internal system. However, with the swift advent of technologies such as web services, agent-base system, the final contexts that are presented to the user on client devices may include content from a number of different heterogeneous data sources. This will certainly have the fundamental effect on the way the whole system is to be built.

Our research in this chapter is different from some of the work that is being done, in that:

(i) Our research is focused on the separation of web content (and web user interface characteristics) from the web content markups and embedded and transformation languages such as HTML, JavaScripts etc.

(ii) We also focused on providing a unified representation of both web content and web user interface description using the LVM approach.

(iii) Also, we simplify the web content description by using XML (and XML Schema) in the LVM, without the need for complex ontology bases or metadata mark-ups.

(iv) We provide a top-down, conceptual model driven design methodology (using LVM framework) and a (XML) storage model (by using views in LVM) for designing and deploying websites (and web portals).

10.5 Example Case Study

To demonstrate our research *x*Web/*x*Portal and the related concepts described in this chapter, we gradually build and present a case study example as described below. It is a website development for our research group's website, **eXel** (EXEL-Lab 2004).

As an example of the research described in this chapter, we developed our research group's website as a simple *x*Web (R.Rajugan et al. 2005a) and a simple *x*Portal (Gardner et al. 2004) system. It is a simple website used as an information source (research collaboration, news, publication list, announcement of seminars and of call-for-papers (CFP), etc.), for public relations (PR), references (for members and their students/collaborators) and for collaborative work (with other research and industry partners/entities). In its simplest form it has four main end user groups, namely:

(i) visitors (any user/agent visiting site),

(ii) collaborative partners,

(iii) members, and

(iv) administrators.

This end user hierarchy is shown in Fig. 10.1. Each group has some predefined privileges in regards to accessing the eXel website, the visitor being the least privileged group and the administrator (here we assume that the administrators are members of the research group) being the highest privileged group. Each group uses a predefined web portal (and associated portlets), namely those shown in Fig. 10.8:

(i) open/generic web portal,

(ii) member-portal,

(iii) collaborator-portal, and

(iv) admin-portal.

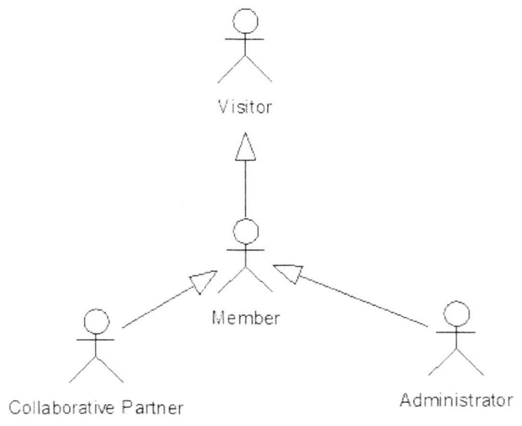


Figure 10.1: eXel web user hierarchy (UML)

A use-case model of this is shown in Fig. 10.4 – 10.5. In a related work, in the context of e-Sol, in (Elizabeth Chang et al. 2003; Elizabeth Chang et al. 2001; ITEC 2002) we discussed in detail the communities (open, closed and locked) based on user access privileges for distributed information systems, such as websites and web portals. Here we employ the same access control for the *x*Portal system, where the member and administrator groups belong to the locked community, whilst the collaborative partner group belongs to a closed community. The visitors (and other web users) belong to open community, where they mainly have view privileges only. The open-portal is the typical web view for all visitors of the site. Its main use is to display public information without access to add, edit or delete permissions. The collaborator portal is for approved collaborative partners of the eXel site (and other participating research groups and/or individual researchers), where they have some added privileges (update publication list, download selected publications, etc) than visitors. Each member of the research group has their own portal and is allowed to view, edit and delete their own work.

In the following sections, we will gradually elaborate on this case study to illustrate our concepts of this chapter. A detailed use-case model of this case study is given in Fig. 10.4 and Fig. 10.5.

It should be noted that, in another related work, we developed a LVM driven User Access Control (UAC) middleware (Steele, Gardner, R.Rajugan et al. 2005) for an e-Health XML repository (Steele, Gardner et al. 2005a, 2005b). But this work, though related to the LVM, is beyond the scope of this research and the UAC mechanism proposed is different from the one described above. Thus, it should not be confused with the research described in this chapter.

10.6 EXtensible Web (xWeb)

In this section, we briefly discuss the major components of the *x*Web (Fig. 10.2), its purpose and the implementation model of the components. *x*Web is composed of three logical components namely, the XML repository, XML view repository and the *x*Web page server. The context diagram of the *x*Web is given Fig. 10.2 and the dependencies between the main components are shown in Fig. 10.3. The *x*Web and *x*Portal components are discussed in the following sections.

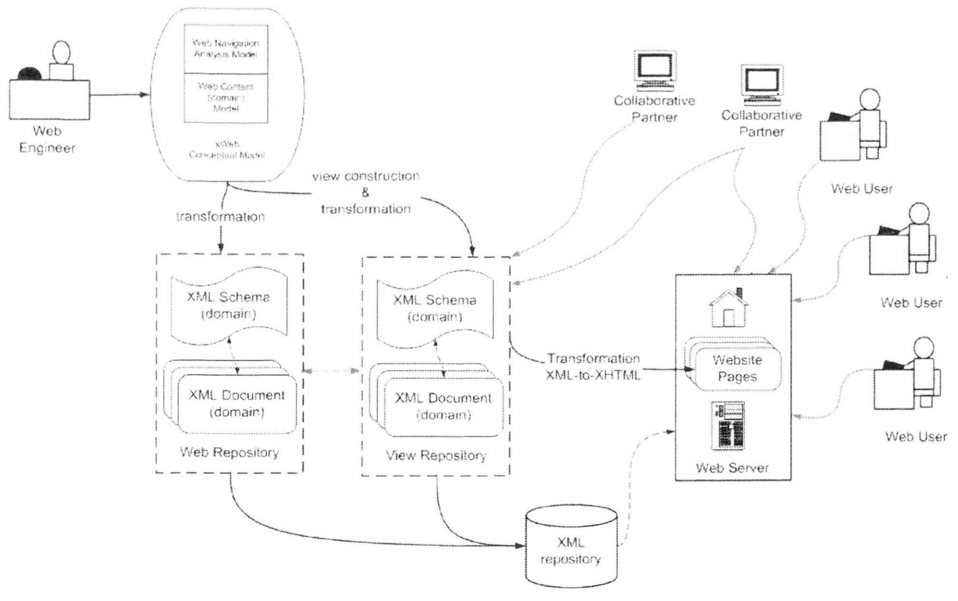


Figure 10.2: *x*Web context diagram

10.6.1 Web (XML) Repository:

The web repository (WebRepos) hosts the website contents (site metadata, data semantics content, web user interface definitions, etc) in analogous to an *x*FACT described in Chapter 8. It is generic XML encoded textual format (with associated schema). The repository is composed of a descriptive, semantically rich repository (XML) schema and an XML document that stores the web contents. Therefore, the

XML repository serves as an organized web information source for building, hosting and distributing web data for the intend use. In simple terms, it replicates a simple yet generic XML-based database management system (DBMS) hosting web contents.

The modelling and transformation of the web content in the WebRepos is analogous to the modelling and transformation of domain models as described in (Feng, Chang & Dillon 2003). WebRepos provides three levels of abstraction and the schemata transformation is done using the approaches described in:

(i) (Feng, Chang & Dillon 2003; Xiaou et al. 2001a; Xiaou et al. 2001b), if the conceptual model is represented using UML, or

(ii) (Feng, Chang & Dillon 2002)if the conceptual model is represented in XSemantic nets.

Storing and maintaining web content in such a repository reduces structural ambiguity among web content, yet maintains semantic richness of each individual web object (this in contrast to hosting web content as relational data). Some of the perceived benefits of such a repository include but are not limited to:

(i) generic, yet semantically descriptive web content repository (a direct result of using XML and XML Schema for content description;

(ii) semantically richer than relational and/or HTML counter parts where schema descriptions are limited and/or optional;

(iii) since it is in native XML format, the web content dissemination (such as publication data in our motivating example website) among collaborators and/or stakeholders are easier than using propriety messaging formats;

(iv) based on native XML technology, thus data is descriptive and supports heterogeneous web structures; and

(v) since the web content is independent from WUI objects and/or constraints (in contrast to HTML data), the data along with the WUI definitions (not technology specific presentation layers) are readily distributable and re-usable in other applications such as web portal generation and collaboration.

In the case of collaborative website engineering, using a XML repository helps in:

(i) keeping the content generic yet semantically descriptive;

(ii) it is XML Schema driven, therefore no need of additional semantic and or schema descriptions and/or mapping at the source and the target in the case of distributing the web content between collaborative partners

(iii) text based (XML) data thus provide support for Unicode & multilingual situations; and

(iv) it keeps the captured web user interface (WUI) definitions independent and free of the web content, presentation markups, structure and format.

As for the physical storage model for the repository, it is readily implemented and hosted in any data storage technology that provides support for native XML documents (and schema) manipulations (from XML-enabled databases servers or native XML DBs).

10.6.2 View (XML) Repository:

The view repository (ViewRepos) is a storehouse of LVM view definitions, associated schemas and materialized instances. Here, a LVM view definition corresponds to a web page representing the web content and its associated structures.

Typically, as stated earlier, at the conceptual level (and also at the other levels of abstraction) a web page in the *x*Web model corresponds to a conceptual view definition constructed in the predefined context of "*web-page*". In addition, all the conceptual views constructed have the same view structure (or template) as shown in Fig. 10.7.

The conceptual view hierarchy ViewRepos is organized in a hierarchical manner that loosely resembles the given website (site map, i.e. the root view/node closely resembles the classical index.html page). Also, most of the conceptual views in the view repository are materialized and periodically updated to reflect the changes and/or updates in the web content (or web server). In theory, the view repository may be considered as an "external schema", analogous to the traditional ANSI/SPAC three schema relational database architecture.

It should be noted that both WebRepos and ViewRepos provide three levels of abstraction, namely conceptual, logical and document levels. The logical model is *generated* by schemata transformation (such as discussed in Chapters 6 and 7) and in the case of document level, the WebRepos is populated with web contents (usually XML documents) and in the case of ViewRepos, document views are instantiated (and later materialised), again resulting in XML documents that are valid against the corresponding schemas generated at the logical level.

10.6.3 Page Sever:

The page server is usually a (classical) web server, serving clients with X/HTML pages. The XHTML pages are generated (usually not done on-fly or in real-time) by applying XSL transformation using the stylesheet definitions (the stylesheet definitions are one of the result of the web navigation model(Gardner 2006) analysis) to the materialized document views that are stored in the view repository (in batch mode, depending on the web content type).

The main advantage of using XHTML-based pages to build screens is performance and compatibility. Other perceived advances of using XHTML-based screens include:

(i) no propriety standards (classical HTML server pages) or browsers needed to view the pages (though originally the web content is based in an XML encoded format);

(ii) easy to implement and maintain (server/client technologies);

(iii) no new scripts and/or web languages embedded into the web pages; and

(iv) no middleware and/or application servers are needed.

A detailed discussion of such transformation (XML view to XHTML) can be found in (Gardner et al. 2004) in the context of web portals.

10.6.4 Portal (XML) Repository:

Similar to the case of view repository, the portal repository is a storehouse of LVM view definitions that are defined over the view definitions of the view repository, given in the predefined context of "*portal*". Here, each conceptual view definition corresponds to a portlet and has a fixed structure (as shown Fig. 10.8) similar to the conceptual views in the view repository. Again, usually the conceptual views are materialized and stored.

10.6.5 Portal Page Sever:

The portal page server is the logical grouping (or separation) of the page server part that hosts the portal definitions.

10.7 xWeb and xPortal Semantics

In this section, we present some of the formal semantics associated with the *x*Web and *x*Portal (without the web user interface and navigation components). Since the proposed model is driven by LVM, some of the concepts and definitions are extensions and/or elaboration of the concepts and definitions presented in Chapter 5.

10.7.1 xWeb Semantics

As described earlier, the uniqueness of *x*Web, in comparisons with other LVM driven applications described in Chapters 8 and 9, is the predefined set of objects, relationship and constraints for both the context and the conceptual views.

Let $xWeb_{item}$ denote an *x*Web item (or element). *x*Web items are the smallest building blocks of the *x*Web model. Let wC_{ontent} denote the web content and wU_{task} denote the web user tasks in an *x*Web item, respectively.

Definition 10.1. An *x*Web item $xWeb_{item}$ is defined as an *ordered* pair such that, it is composed of two elements, the (web) user tasks wU_{task} that correspond to the model elements of the web navigation model and the web content wC_{ontent} that corresponds to the web content semantics and the representational model elements of an *x*Web item.

$$xWeb_{item} \equiv < wU_{task}, wC_{ontent} > \qquad \textbf{(10.1)}$$

It should be noted that there exists a strong coupling between the web content and the user tasks, as web user tasks determine the access (access control, access level, etc.), order (the order in which the web content is displayed to the user, as part of the website hierarchy) and presentation (typical web page presentation elements such as text size, font, colour, etc.) properties of each element of the web content. It can be said that web user tasks determine web content properties. This can be shown as:

$$wU_{task} \mapsto wC_{ontent} \qquad \textbf{(10.2)}$$

Conversely, the user task element definitions are *represented* within the corresponding web content, as part of its properties, analogous to an external message body definition, i.e. a message that triggers a state change in one or more external object(s).

As part of *x*Web (and *x*Portal) research, in this chapter, we only present definitions and semantics of the web content as an application of the LVM. The definitions and semantics of the web user tasks are outside the scope of this chapter and is part of another PhD thesis in (Gardner 2006), titled *"Declarative Service Perspective Web Interface Design Approach – A Structural and Dynamic Modelling Framework"*. We direct the interested reader to our combined work in (Gardner et al. 2004; R.Rajugan et al. 2005a, 2005b) or for more detailed discussion of the web user tasks (and web navigation model) in (Gardner 2006). We continue our discussion on web content below.

Let xP_{age} denotes a set of pages in *x*Web.

Definition 10.2. Web content wC_{ontent} in the *x*Web item $xWeb_{item}$ is a set of *x*Web pages xP_{age}.

$$wC_{ontent} = \{xP_{age}\} \qquad \textbf{(10.3)}$$

As we previously defined in Chapter 5, let ζ_{ontext} denote a context and V^{c}_{iew} denote a set of conceptual views defined over the context. Also, ζ^{wp}_{ontext} be a predefined context of the "web-page", such that $\zeta^{wp}_{ontext} \in \zeta_{ontext}$.

Definition 10.3. An *x*Web page xP_{age} is a conceptual view defined in the context of "web- page" in the web repository.

$$wC_{ontent} = \{(xP_{age})(\zeta^{wp}_{ontext}) \mid (\forall xP_{age})(xP_{age} \in V^{c}_{iew}) \wedge \zeta^{wp}_{ontext} \in \zeta_{ontext}\} \qquad \textbf{(10.4)}$$

Let xp be an *x*Web page such that $xp \in xP_{age}$. Since xP_{age} is a conceptual view V^{c}_{iew} in the given context of ζ^{wp}_{ontext}, using definition 5.6 in Chapter 5, we can define xp such that:

Definition 10.4. An *x*Web page xp is a conceptual view such that xp is a 4-ary tuple of $xp_{name}, xp_{obj}, xp_{rel}$ and $xp_{constraint}$, where xp_{name} is the name of the *x*Web page xp, xp_{obj} is a set of objects in xp, xp_{rel} is a set of object relationships in xp, and $xp_{constraint}$ is a set of constraints associated with xp_{obj} and xp_{rel} in xp.

$$xp = (xp_{name}, xp_{obj}, xp_{rel}, xp_{constraint}) \qquad \textbf{(10.5)}$$

Remarks 1:

(i) In xWeb, as stated before, the context is always predefined in contrast to other applications of the LVM described in Chapters 8 and 9.

(ii) Another notable aspect here is that, the objects (xp_{obj}) in xP_{age} are also preset, to a template like structure. This also implies that the object relationships (xp_{rel}) and the constraints ($xp_{constraint}$) are also predefined.

(iii) We can show that xp is valid against the given context ζ^{wp}_{ontext} using the definition given in Chapter 5, definition 5.3.

(iv) The template structure is shown in Fig. 10.7, together with the predefined, template structure of the context, "web-pages" (ζ^{wp}_{ontext}) in Fig. 10.6.

10.7.2 xPortal Semantics

In regards to *x*Portal semantics, it may be considered as a layer on top the *x*Web, with aggregated *x*Web model elements. Thus, the semantics are very similar to the *x*Web.

In *x*Portal, similar to *x*Web items, the smallest building blocks of the *x*Portal model is the concept of *x*Portal item. Let $xPortal_{item}$ denote an *x*Portal item. Let pC_{ontent} denote the portal content and pU_{task} denote the portal user tasks in an *x*Portlet, respectively. Thus, similar to *x*Web item, we can define an *x*Portal item as:

Definition 10.5. An *x*Portal item $xPortal_{item}$ is defined as an *ordered* pair such that, it is composed two elements, the portal user tasks pU_{task} that correspond to the model elements of the web navigation model and the portal web content pC_{ontent} that corresponds to the portal content semantics and the representational model elements of an *x*Portal item.

$$xPortal_{item} \equiv < pU_{task}, pC_{ontent} > \qquad \textbf{(10.6)}$$

Again, similar to the *x*Web case, we present only definitions and semantics of the portal web content as it is the application of the LVM. The definitions and semantics of the portal user tasks (pU_{task}) are outside the scope of this chapter and more information can be found in (Gardner et al. 2004; R.Rajugan et al. 2005b).

It should be note that an *x*Portal item is a superset of *x*Web items or more. Therefore,

$$xWeb_{item} \subseteq xPortal_{item} \qquad \textbf{(10.7)}$$

Let xP_{ortlet} denotes a set of portlets in the *x*Portal. This is analogous to the portlets in the traditional web portals. Also, let ζ^{p}_{ontext} be a predefined context of "portal" such that $\zeta^{p}_{ontext} \in \zeta_{ontext}$.

Definition 10.6. An *x*Portlet xP_{ortlet} is a conceptual view defined in the context of "portal" in the view repository.

Let xP^{p} denotes an *x*Portlet ($xP^{p} \in xP_{ortlet}$). Therefore, similar to equation 10.5, using equation 5.6 in Chapter 5, we can show that, an *x*Portlet xP^{p}:

$$xP^{p} = (xP^{p}{}_{name}, xP^{p}{}_{obj}, xP^{p}{}_{rel}, xP^{p}{}_{constraint}) \qquad \textbf{(10.8)}$$

Remarks 2:

(i) Similar to the case of *x*Web, the objects ($xP^{p}{}_{obj}$) in xP^{p} are *preset*, to a template like structure. This also implies that the object relationships ($xP^{p}{}_{rel}$) and the constraints ($xP^{p}{}_{constraint}$) are also predefined.

(ii) Also, xP^{p} is valid against the given context ζ^{p}_{ontext} similar to the definition given in Chapter 5, definition 5.3.

(iii) The template structure is shown in Fig. 10.8, together with the predefined, template structure of the context, "portal" (ζ^{p}_{ontext}).

Let xP_{ortal} denotes a set of web portals in *x*Portal model.

Definition 10.7. An *x*Portal xP_{ortal} in *x*Portal is a container for a set of semantically related portlets in the *x*Portal xP_{ortlet} that are constructed in the predefined context of "portal" ζ^{p}_{ontext}.

$$xP_{ortal} = \{(xP^{p})(\zeta^{p}_{ontext}) \mid (\forall xP^{p})(xP^{p} \in xP_{ortlet} \wedge \zeta^{p}_{ontext} \in \zeta_{ontext})\} \qquad \textbf{(10.9)}$$

Thus, we can define the portal web content pC_{ontent} as:

$$pC_{ontent} = \{(xP)(\zeta^{p}_{ontext}) \mid (\forall xP)(xP \in xP_{ortal}) \wedge \zeta^{p}_{ontext} \in \zeta_{ontext}\} \qquad \textbf{(10.10)}$$

$$\begin{aligned} &pC_{ontent} = \\ &\{(xP)(xP^{p})(\zeta^{p}_{ontext}) \mid (\forall xP^{p}) \\ &\qquad ((xP^{p} \in xP_{ortlet} \wedge \zeta^{p}_{ontext} \in \zeta_{ontext}) \wedge xP^{p} \in xP) \\ &\wedge (\forall xP)(xP \in xP_{ortal})\} \end{aligned} \qquad \textbf{(10.11)}$$

In the case of logical model semantics in the *x*Web (and *x*Portal) model, since its concepts are based on the LVM, analogous to the logical view definitions, we can derive the definitions as shown below.

The logical model of the *x*Web page may be written using the definition 5.7 as:

$$xp^{s} = (xp^{s}_{name}, xp^{s}_{simpleType}, xp^{s}_{complexType}, xp^{s}_{constraints}) \qquad \textbf{(10.12)}$$

where xp^s denotes the logical (or schema) model of the *x*Web page. Also $xp^s \in V^s_{iew}$. Also, though not presented here, the schemata transformation functions and rules stated in Chapter 5 are valid here as well.

Similarly, in the case of *x*Portles:

$$xP^{p^s} = (xP^{p^s}_{name}, xP^{p^s}_{simpleType}, xP^{p^s}_{complexType}, xP^{p^s}_{constraints}) \qquad \textbf{(10.13)}$$

where xp^{p^s} denotes the logical (or schema) equivalent of the model of the xp^p. Also $xp^{p^s} \in V^s_{iew}$.

Similarly, though not shown here, the document (or instance) model of these concepts (and the transformation) may be intuitively deduced from the definitions in Chapter 5 (definitions 5.7 and 5.8). However, it should be noted that, since the context and conceptual views (templates) are predefined, the conceptual operators used in the *x*Web model are usually a Selection-Projection-Join (SPJ) type.

The following section presents some of the modelling issues and steps associated with the *x*Web and *x*Portal.

10.8 A Practical Walkthrough of the xWeb Design Methodology

The *x*Web design methodology provides three levels of abstraction, analogous to the LVM, since *x*Web is based on the LVM framework. The levels of abstraction include: (a) conceptual, (b) logical or schema and (c) document levels. Thus, it can be said that the *x*Web methodology provides a 3-step design methodology to engineer web contents (and interfaces), which is proposed in the thesis (Gardner 2006). We modify it here, concentrating on the prospects of web page design as views of XML repositories.

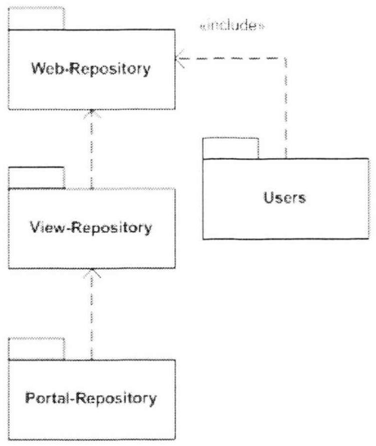


Figure 10.3: *x*Web/*x*Portal conceptual level dependencies (context diagram)

At the conceptual level (Fig. 10.3): (Gardner 2006):

(1) design the web site and its semantics (such as site layout, structure, data, user access control) using a generic UML model, which serves as the conceptual model for the web (XML) repository for a given website;

(2) develop abstract user interface definitions using abstract (web) user interface (AUI) objects or user perspectives (Gardner 2006; Gardner et al. 2004); and

(3) derive conceptual views to build the web pages (i.e. a web page construct corresponds to one (or more) conceptual view/(s)) and web portals (view of a view/ aggregate view). This serves as the conceptual model for the web (view) repository.

At the logical level, we:

(1) transform the web (XML) repository captured in UML model to logical model;

(2) transform AUI objects to web user interface (WUI) definitions such as stylesheet definitions; and

(3) transform conceptual views to logical view (schema) definitions using transformation rules described in Chapter 7.

Finally at the document level, the transformation is threefold:

(1) fill web repository with data the website data (page contents, layouts, resources etc);

(2) instantiate document views (materialized views) and validate against the logical views;

(3) generate XHTML documents using materialized document views and the UI definitions using the corresponding XSLT transformations

Based on the above discussion, we now proceed to show the design steps involved in building an *x*Web based website (and a web portal) using the example case study. The design time methodology has the following steps:

- ***Step 1***: Development of conceptual model of the web domain in question: At the conceptual level, the website domain and data requirements are analysed and represented using UML by applying OO conceptual modelling techniques with extensive use-case analysis. The results achieved for the case study example (i.e. use-case analysis) is given in Figs. 10. 4 - 10.5.

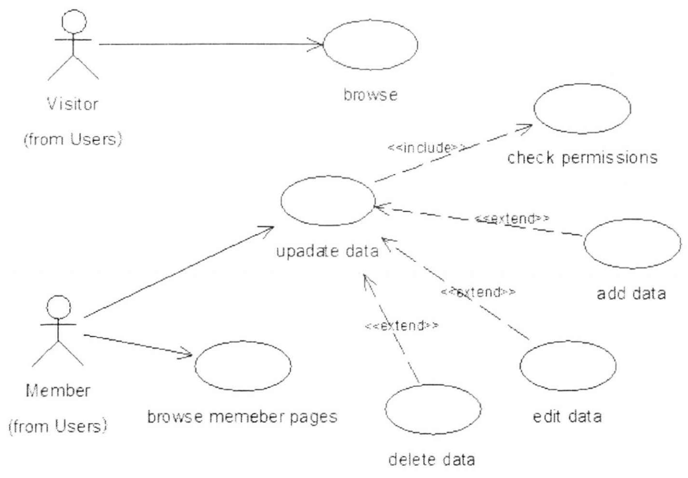


Figure 10.4: Use-case analysis of the example case study (visitor, member)

- ***Step 2***: Also, an extensive web user navigation analysis is done and a web user analysis navigation model is developed as a complementary model to the conceptual model developed in *Step 1*. This includes identification and (graphical) representation of user interaction in the context of web content, including definitions of screens, their components and the navigation links. A detailed discussion of this can be found in (Elizabeth Chang & Dillon 1994; Gardner 2006).

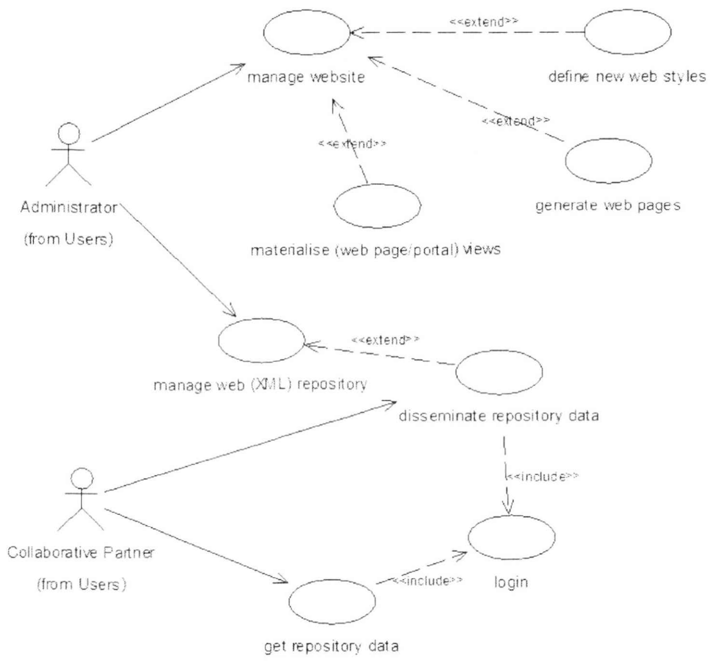


Figure 10.5: Use-case analysis of the case study (administrator, collaborative partner)

- ***Step 3***, Development of conceptual model of the web repository: Based on the results obtained in *Step 1*,(i.e. use-case analysis, web user navigation analysis, etc.), the conceptual model is mapped to the web repository model (template) under three categories namely:
  - (a) web contents,
  - (b) web resources (hypermedia objects host at the website), and
  - (c) user interface definitions (done in step 2).

The model template developed for the web repository is shown in Fig. 10.6.

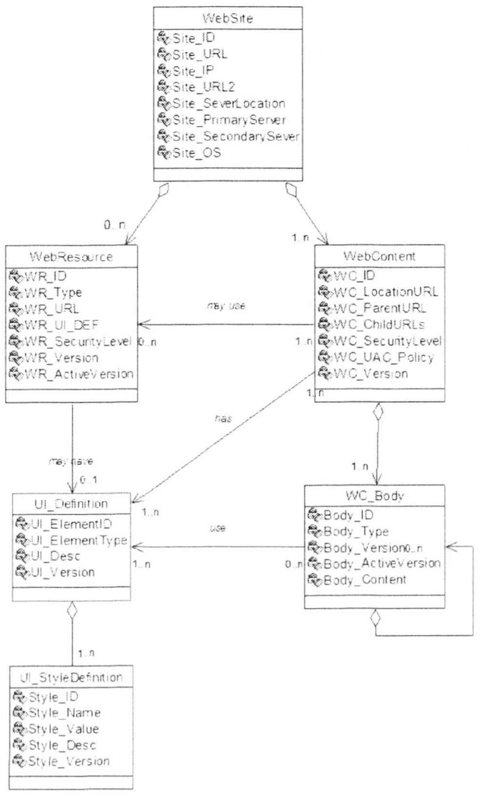


Figure 10.6: Web repository model (conceptual level)

- *Step 4*, Development of *website* conceptual model: here, based on the web repository, *conceptual views* are constructed to satisfy *x*Web pages (i.e. web pages in the *x*Web), which in turn satisfy one or more web user requirement/(s). The conceptual views are constructed from the web repository for the given context of "*web-page*". Fig. 10.7 shows this model template (i.e. *x*Web pages).

- *Step 5*, Development of web portal conceptual model: Here, based on view repository, portal/portlet *conceptual views* (called portlet and portal and represented in UML using `<<Portlet>>` or `<<Portal>>` stereotypes, shown in Fig. 10.8) are constructed to satisfy user/domain requirements. Again, the conceptual views are constructed from view repository in the given context of '*portal*'. Fig. 10.8 shows this model that was developed for the case study example.

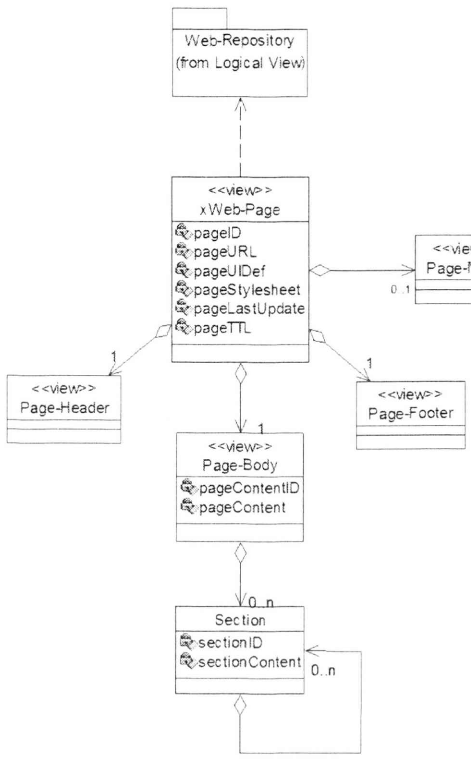


Figure 10.7: View repository model (conceptual level)

- ***Step 6,*** Development of the web repository logical model: Here, the conceptual model of the web repository is mapped to logical level (XML schema) using schemata transformation rules described in Chapters 6 & 7. The transformations are dependent on the modelling language used to represent the web repository. Here, we demonstrate the transformation described in Chapter 7 for mapping conceptual views represented using UML. A graphical representation of the web repository logical model (XML Schema) developed is shown in Fig. 10.9.

- ***Step 7(a)/(b),*** Development of the view repository (and the corresponding portal repository) logical model: The conceptual views defined in Step 4 are mapped to the view repository logical (XML Schema) model. A graphical representation of this (XML Schema) is shown in Fig. 10.10.

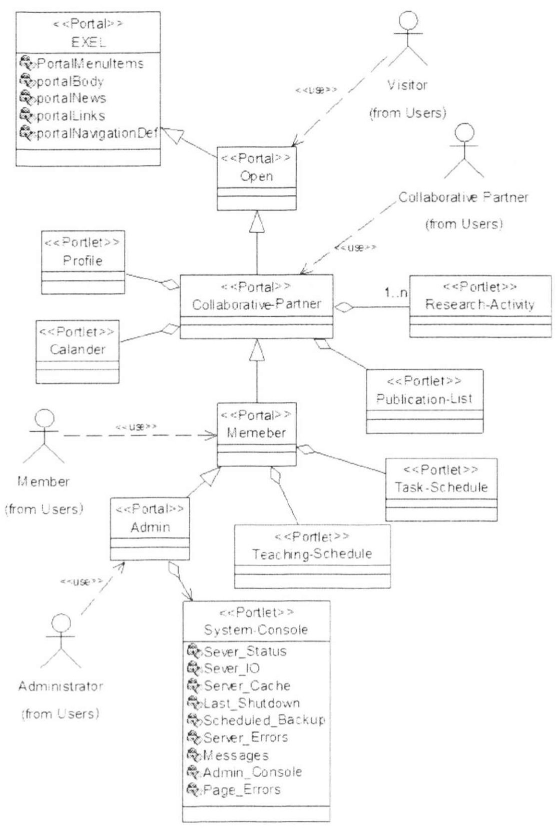


Figure 10.8: Portal repository model (conceptual level)

- ***Step 8,*** Development of the web repository document model: at this stage, the web repository is populated with the web contents. Later, the populated repository document/(s) are validated against the web repository schema generated during Step 6.

- ***Step 9(a),*** Development of the view repository document model: here, the document views are instantiated and validated against the logical level (view schema) constructed in Step 7(a) above. The view instantiation (and materialization) using document view (XQuery) expression or simple XPath or XSLT queries.

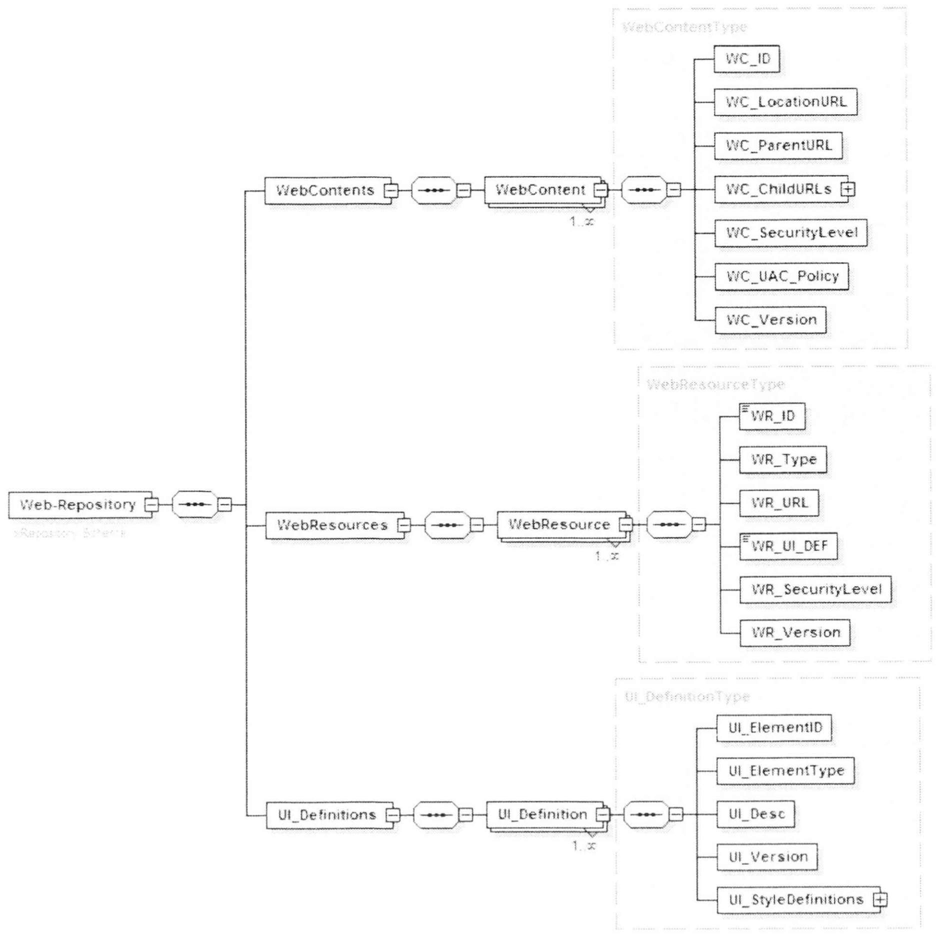


Figure 10.9: Web repository logical model (graphical representation of the resulted XML Schema)

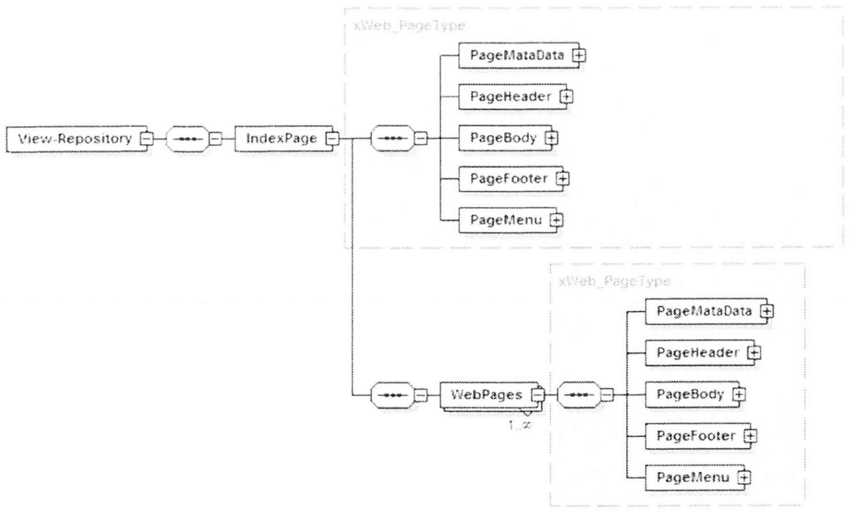


Figure 10.10: View repository logical model (graphical representation of the resulted XML Schema)

(a) ***Step 9(b)***, Development of the portal repository document model: Similar to Step 9(a) above, the task is repeated for portal document views.

(b) ***Step 10(a)***, Development of the *x*Web Pages: here the materialised LVM (XML) views are transformed into XHTML pages using XSLT and stylesheet definitions, thus generating XHTML pages to for the web servers. The main advantage of using XHTML based pages to build screens is performance and compatibility. Example xWeb pages are shown in Figs. 10.11 – 10.13

Other perceived advances of using HTML based screens include:

(i) no propriety standards (classical HTML server pages) or browsers needed to view the pages (though originally the web content is based in an XML encoded format);

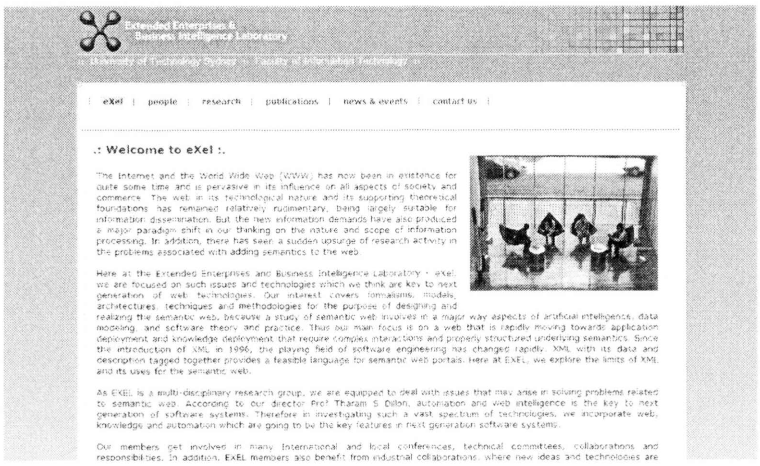
(ii) easy to implement and maintain (server/client technologies);

(iii) no new scripts and/or web languages embedded into the web pages; and

(iv) no middleware and/or servers are needed.

The generation of the resulting web page (in XHTML) is enabled by the XSL transformation, which simply converts the document views (in association with its schema) into an XHTML pages using the presentation (and the resource) related information stored in the Cascade Style Sheets (CSS). An example of such a transformation is shown below in step 10(b) in the context of *x*Portals.

- ***Step 10(b),*** Development of the xPortal (XHTML) Pages: Step 10(a) is repeated here for the web portal/portlets.


Extended Enterprises &
Business Intelligence Laboratory
:: University of Technology Sydney :: Faculty of Information Technology ::
| eXel | people | research | publications | news & events | contact us |

.: Welcome to eXel :.

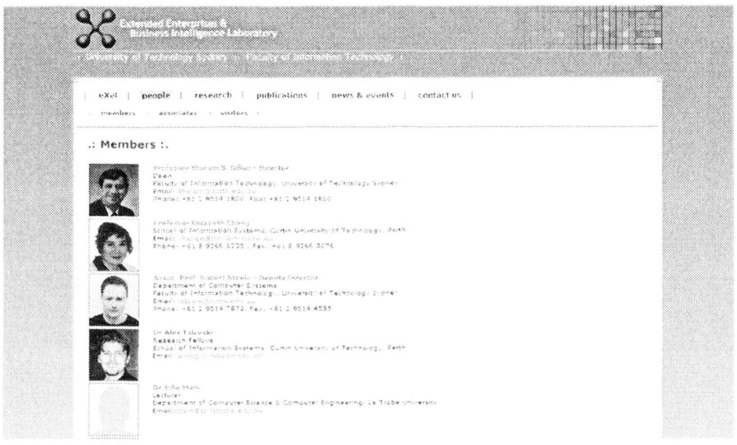
The Internet and the World Wide Web (WWW) has now been in existence for quite some time and is pervasive in its influence on all aspects of society and commerce. The web in its technological nature and its supporting theoretical foundations has remained relatively rudimentary, being largely suitable for information dissemination. But the new information demands have also produced a major paradigm shift in our thinking on the nature and scope of information processing. In addition, there has seen a sudden upsurge of research activity in the problems associated with adding semantics to the web.

Here at the Extended Enterprises and Business Intelligence Laboratory - eXel, we are focused on such issues and technologies which we think are key to next generation of web technologies. Our interest covers formalisms, models, architectures, techniques and methodologies for the purpose of designing and realizing the semantic web, because a study of semantic web involves in a major way aspects of artificial intelligence, data modeling, and software theory and practice. Thus our main focus is on a web that is rapidly moving towards application deployment and knowledge deployment that require complex interactions and properly structured underlying semantics. Since the introduction of XML in 1996, the playing field of software engineering has changed rapidly. XML with its data and description tagged together provides a feasible language for semantic web portals. Here at EXEL, we explore the limits of XML and its uses for the semantic web.

As EXEL is a multi-disciplinary research group, we are equipped to deal with issues that may arise in solving problems related to semantic web. According to our director Prof Tharam S Dillon, automation and web intelligence is the key to next generation of software systems. Therefore in investigating such a vast spectrum of technologies, we incorporate web, knowledge and automation which are going to be the key features in next generation software systems.

Our members get involved in many International and local conferences, technical committees, collaborations and responsibilities. In addition, EXEL members also benefit from industrial collaborations, where new ideas and technologies are

Figure 10.11: *x*Web page example (http://exel.it.uts.edu.au/index.html)


Extended Enterprises &
Business Intelligence Laboratory
:: University of Technology Sydney :: Faculty of Information Technology ::
| eXel | people | research | publications | news & events | contact us |
: members : associates : visitors :

.: Members :.

Professor Tharam S. Dillon - Director
Dean
Faculty of Information Technology, University of Technology Sydney
Email: [illegible]
Phone: +61 2 9514 1800, Fax: +61 2 9514 1816

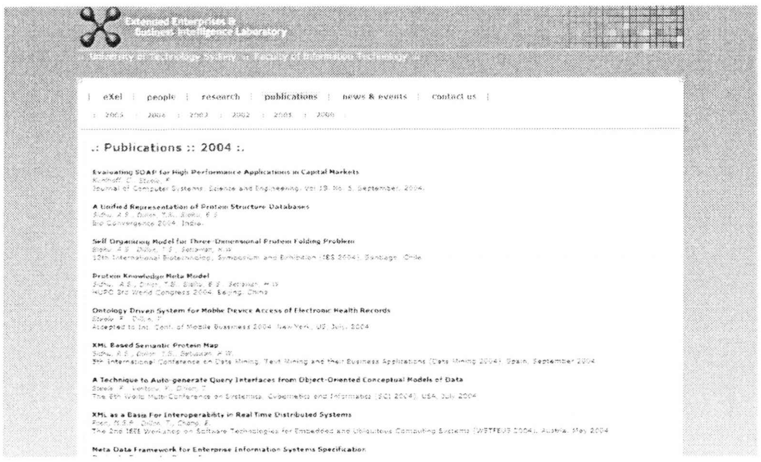
Professor Elizabeth Chang
School of Information Systems, Curtin University of Technology, Perth
Email: [illegible]
Phone: +61 8 9266 1??? , Fax: +61 8 9266 3076

Assoc. Prof. Robert Steele - Deputy Director
Department of Computer Systems
Faculty of Information Technology, University of Technology Sydney
Email: [illegible]
Phone: +61 2 9514 7872, Fax: +61 2 9514 4535

Dr Alex Talevski
Research Fellow
School of Information Systems, Curtin University of Technology, Perth
Email: [illegible]

Dr [illegible]
Lecturer
Department of Computer Science & Computer Engineering, La Trobe University
Email: [illegible]

Figure 10.12: *x*Web page example - member list (http://exel.it.uts.edu.au/people.php)


Extended Enterprises &
Business Intelligence Laboratory
:: University of Technology Sydney :: Faculty of Information Technology ::
| eXel | people | research | publications | news & events | contact us |
: 2005 : 2004 : 2003 : 2002 : 2001 : 2000 :

.: Publications :: 2004 :.

Evaluating SOAP for High Performance Applications in Capital Markets
Kohlhoff, C., Steele, R.
Journal of Computer Systems, Science and Engineering, Vol 19, No. 5, September, 2004.

A Unified Representation of Protein Structure Databases
Sidhu, A.S., Dillon, T.S., Sidhu, B.S.
Bio Convergence 2004, India.

Self Organizing Model for Three-Dimensional Protein Folding Problem
Sidhu, A.S., Dillon, T.S., Setiawan, H.W.
12th International Biotechnology Symposium and Exhibition (IBS 2004), Santiago, Chile

Protein Knowledge Meta Model
Sidhu, A.S., Dillon, T.S., Sidhu, B.S., Setiawan, H.W.
HUPO 3rd World Congress 2004, Beijing, China

Ontology Driven System for Mobile Device Access of Electronic Health Records
Steele, R., Dillon, T.
Accepted to Int. Conf. of Mobile Business 2004, New York, US, July, 2004

XML Based Semantic Protein Map
Sidhu, A.S., Dillon, T.S., Setiawan, H.W.
5th International Conference on Data Mining, Text Mining and their Business Applications (Data Mining 2004), Spain, September 2004

A Technique to Auto-generate Query Interfaces from Object-Oriented Conceptual Models of Data
Steele, R., Ventsov, Y., Dillon, T.
The 8th World Multi-Conference on Systemics, Cybernetics and Informatics (SCI 2004), USA, July 2004

XML as a Basis For Interoperability in Real Time Distributed Systems
Poon, N.S.P., Dillon, T., Chang, E.
The 2nd IEEE Workshop on Software Technologies for Embedded and Ubiquitous Computing Systems (WSTFEUS 2004), Austria, May 2004

Meta Data Framework for Enterprise Information Systems Specification

Figure 10.13: *x*Web page example - publications (http://exel.it.uts.edu.au/publications.php)

For example, a section of the document view instance that represents the portlet of the member profile is shown in the Code Listing 10.1 below. It should be noted that this is only a part of a larger XML document (Note: The XML view schema is not shown here.) which resulted from the view construction process. Each portlet is composed of three sub-structures, namely the Portlet_METADATA, Portlet_Data, and Porlet_Action. A sample *x*Portal page is shown in Fig. 10.14.

```
<!--Further Nesting-->
<WebPortlets>
    <Portlet UAC_access="all" type="profile">
        <Portlet_METADATA>
            <Portlet_ID><!-- data --></Portlet_ID>
            <Portal_Type><!-- data --></Portal_Type>
            <Portlet_URL><!-- data --></Portlet_URL>
            <Portal_AccessPort><!-- data --></Portal_AccessPort>
        </Portlet_METADATA>
        <Portlet_Data UI_DEF="profile.css">
            <Profile_ID><!-- data --></Profile_ID>
            <!--Further Nesting-->
        </Portlet_Data>
        <Portlet_Action>
            <Action
        ref="http://exel.it.uts.edu.au/member/profiles?pf=10090277&amp;action=update">
Update</Action>
            <!--Further Nesting-->
        </Portlet_Action>
    </Portlet>
    <Portlet>
        <!--Further Nesting-->
    </Portlet>
    <!--Further Nesting-->
</Portlets>
   <!--Further Nesting-->
```

Code Listing 10.1: Sections of document view instance (xPortal)

The important issue here is the utilization of CSS entries in capturing the abstract (web) user interface definitions (from the conceptual level) at the logical level, which is supplied by the UI_DEF attribute for each portlet. The transformation process here will make no interpretation of the semantics of the data, as the semantic of the data is provided by the LVM views during the modelling and transformation process. The CSS definition for the profile portlet (Code Listing 10.2) is captured in the profile.css file. While it may contain other style information, the important selector in this case is the CSS class definition of label and content.

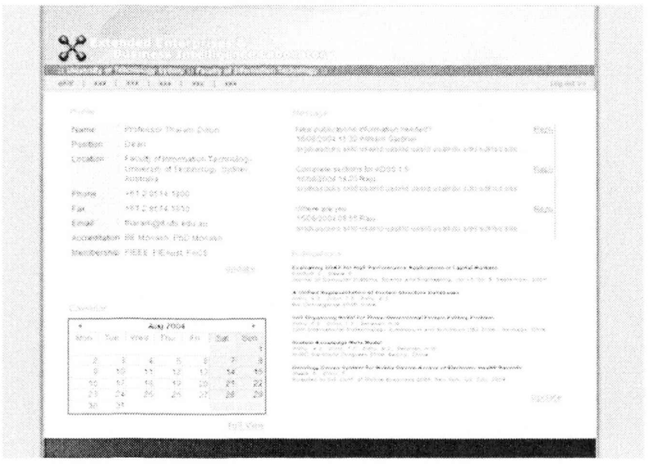
```
.label{
   font-family: Verdana, Arial, Helvetica, sans-serif;
   background-color: #E1E1A8;
   font-weight: bold;
   width: 28;
   font-size: 1em;
   }
   .content{
    font-family: Verdana, Arial, Helvetica, sans-serif;
    font-size: 1em;
  }
```

Code Listing 10.2: Section of CSS definition in profile.css

In Code Listing 10.3, the sections of XSLT coding that represent two elements are shown. It combines presentational information from the CSS and data from the document view instances. Note here that the use of any numeric value in the CSS definition should be in reference unit (e.g. for font size, the use of em). While each portlet is allowed to have its individual CSS file, the generic portals do have a general style of a CSS file. The resulting XHTML page presentation (in a web browser) of the Member portal (under development in the *x*Portal system) is shown in Fig. 10.14.

```
<tr>
  <td class="label">Name</td>
  <td class="content">
       <xsl:value-of select="Profile_Name"/>
  </td>
</tr>
<tr>
  <td class="label">Position</td>
  <td class="content">
        <xsl:value-of select="Profile_Position"/>
  </td>
</tr>
```

Code Listing 10.3: XSLT code (sample)



Name: Professor Tharam Dillon
Position: Dean
Location: Faculty of Information Technology, University of Technology, Sydney, Australia
Phone: +61 2 9514 1800
Fax: +61 2 9514 1810
Email: tharam@it.uts.edu.au
Accreditation: BE Monash, PhD Monash
Membership: FIEEE, FIEAust, FACS

Aug 2004

Figure 10.14: *x*Portal example - member profile (under development)

10.9 Conclusion

In this chapter, we presented an intuitive approach to model and design web pages and web portals using the views in the LVM (in combination with a web navigation analysis model). First, we presented some web engineering concepts, followed by some related work done in the area of conceptual modelling and website (and portal) development. Then we described our research, *x*Web (and *x*Portal) its unique characteristics and followed by some of the formal semantics and definitions associated the *x*Web and the *x*Portal. Later, we presented a practical walkthrough using an illustrative case study example, including design steps, transformation and deployment issues. It should be noted that, some of the implementation issues and problems associated with the *x*Web and *x*Portal are still pending or ongoing work due to time constraints and will be addressed as part of the future work section.

This chapter concludes the applications of the LVM, thus completing this thesis. In the following chapter, we provide recapitulation of this thesis and present some future research directions and work associated with this thesis.

References

Abiteboul, S, Amann, B, Cluet, S, Eyal, A, Mignet, L & Milo, T 1999, 'Active Views for Electronic Commerce', *Proceedings of the 25th International Conference on VLDB*, Edinburgh, Scotland, pp. 138-49.

Abiteboul, S, Benjelloun, O, Manolescu, I, Milo, T & Weber, R 2002, 'Active XML: A Data-Centric Perspective on Web Services', *BDA*.

Aragones, A & Hart-Davidson, W 2002, 'Why, When and How do Users Customize Web Portals?' *Proceedings of IEEE International Professional Communication Conference (IPCC '02)*, vol. A. Aragones, W. Hart-Davidson, "Why, When and How do Users Customize Web Portels?"; Proceedings of IEEE International Professional Communication Conference, 2002. IPCC 2002.

Bellas, F, Fernández, D & Muiño, A 2004, 'A Flexible Framework for Engineering "My" Portals', *13th International World Wide Web Conference, WWW2004*.

Booch, G 1993, *Object-oriented analysis and design with applications*, 2nd edn, The Benjamin/Cummings series in object-oriented software engineering., Benjamin/Cummings Pub. Co.; Addison-Wesley, Redwood City, Calif. Reading, Mass.

Ceri, S, Fraternali, P & Bongio, A 2000, 'Web Modeling Language (WebML): a Modeling Language for Designing Web Site', *9th International World Wide Web Conference, WWW2000*.

Chan, S, Dillon, TS & Siu, A 2002, 'Applying a mediator architecture employing XML to retailing Inventory Control', *The Journal of Systems and Software*, vol. 60, pp. 239-48

Chang, E 1996, 'Object Oriented User Interface Design and Usability Evaluation', Doctor of Philosophy (Ph.D) thesis, La Trobe University, Melbourne, Australia.

Chang, E, Dillon, T, Gardner, W, Talevski, A, R.Rajugan & Kapnoullas, T 2003, 'A Virtual Logistics Network and an e-Hub as a Competitive Approach for Small to Medium Size Companies', *2nd International Human.Society@Internet Conference*, Springer-Verlag, Seoul, Korea, pp. 265-71.

Chang, E & Dillon, TS 1994, 'Integration of User Interfaces with Application Software and Databases Through the Use of Perspectives', *1st International Conference on Object-Role Modeling (ORM '94)*, Australia, pp. 4-6.

Chang, E & Dillon, TS 1999, 'Software Development Methodology and Structure of a Class of Multimedia Web Based Systems', *2nd Asia-Pacific Web Conference - Technologies and Applications for the New Millennium, APWeb 99*, Hong Kong.

Chang, E, Gardner, W, Talevski, A, Gautama, E, R.Rajugan, Kapnoullas, T & Satter, S 2001, 'Virtual Collaborative Logistics and B2B e-Commerce', *e-Business Conference*, Duxon Wellington, NZ.

Conallen, J 1999, 'Modeling Web Application Architectures with UML', *Communictaions of the ACM*, vol. 42, no. 10, October.

Conallen, J 2003, *Building Web Application with UML*, 2 edn, Addison-Wesley.

Dillon, TS & Tan, PL 1993, *Object-Oriented Conceptual Modeling*, Prentice Hall, Australia.

Do, HH & Rahm, E 2004, 'Flexible integration of molecular-biological annotation data: The genmapper approach', *Proceedings of the 9th International Conference on Extending Database Technology (EDBT '04)*, Heraklion, Crete, Greece.

EXEL-Lab 2004, *Extended Enterprises & Business Intelligent Laboratory, (http://exel.it.uts.edu.au/)*, http://exel.it.uts.edu.au/, http://exel.it.uts.edu.au/.

Feng, L, Chang, E & Dillon, TS 2002, 'A Semantic Network-based Design Methodology for XML Documents', *ACM Transactions on Information Systems (TOIS)*, vol. 20, No 4, no. 4, pp. 390 - 421

Feng, L, Chang, E & Dillon, TS 2003, 'Schemata Transformation of Object-Oriented Conceptual Models to XML', *International Journal of Computer Systems Science & Engineering*, vol. 18, No. 1, no. 1, pp. 45-60

Gaedke, M, Segor, C & Gellersen, H-W 2000, 'WCML: paving the way for reuse in object-oriented Web engineering', *Proceedings of the ACM symposium on Applied computing (SIGAPP '00)*, ACM Press, Italy, pp. 748-55.

Gant, JP & Gant, DB 2002, 'Web portal functionality and State government E-service', *Proc.of the 35th Annual Hawaii International Conference on System Sciences (HICSS '02)*, Hawaii.

Gardner, W 2006, 'Declarative Service Perspective Web Interface Design Approach – A Structural and Dynamic Modelling Framework', Doctor of Philosophy (PhD) thesis, University of Technology, Sydney (UTS), Sydney -- *submitted for examination.*

Gardner, W, Chang, E & Dillon, TS 2003a, 'Analysis Model of Web User Interface for Web Applications', *16th International Conference on Software & Systems Engineering and their Applications (ICSSEA '03)*, France.

Gardner, W, Chang, E & Dillon, TS 2003b, 'Two Layer Web User Interface Analysis Framework Using SNN and iFIN', *On The Move to Meaningful Internet Systems 2003: OTM 2003 Workshops HCI-SWWA*, vol. LNCS 2889, pp. 223-34.

Gardner, W, R.Rajugan, Chang, E & Dillon, TS 2004, 'xPortal: XML View Based Web Portal Design', *17th International Conference on Software & Systems Engineering and their Applications (ICSSEA '04)*, Paris, France.

Garzotto, F, Paolini, P & Schwabe, D 1993, 'HDM - A Model-Based Approach to Hypertext Application Design', *ACM Transactions on Information Systems (TOIS)*, vol. 11, no. 1, pp. 1-26

Gruber, TR 1993, 'A Translation Approach to Portable Ontologies', *Knowledge Acquisition*, vol. 5, no. 2, pp. 199-220

Gu, A, Henderson-Sellers, B & Lowe, D 2002, 'Web Modeling Languages: the gap between requirements and current exemplars', *8th Australian World Wide Web Conference, AUSWEB'02.*

ITEC 2002, *iPower Logistics (http://www.logistics.cbs.curtin.edu.au/)*, http://www.logistics.cbs.curtin.edu.au/.

Lei, Y, Motta, E & Domingue, J 2002, 'An Ontology-Driven Approach to Web Site Generation and Maintenance', *Proceedings of 13th International Conference on Knowledge Engineering and Management*, Springer-Verlag, Sigüenza, Spain, pp. 219-34.

Lei, Y, Motta, E & Domingue, J 2004a, 'Modelling Data-Intensive Web Sites with OntoWeaver', *Proceedings of the International Workshop on Web Information Systems Modelling (WISM '04)*, Springer-Verlag, Riga, Latvia.

Lei, Y, Motta, E & Domingue, J 2004b, 'OntoWeaver-S: Supporting the Design of Knowledge Portals', *Proceedings 14th International Conference on Knowledge Engineering and Knowledge Management (EKAW '04)*, Northamptonshire, UK.

OMG-MDA 2003, *The Architecture of Choice for a Changing World®, MDA Guide Version 1.0.1 (http://www.omg.org/mda/)*, OMG, 2005.

R.Rajugan, Gardner, W, Chang, E & Dillon, TS 2005a, 'Designing Websites with EXtensible Web (xWeb) Methodology', *International Journal of Web Information Systems (IJWIS)*, vol. 1(3), no. 1, pp. 179-91, September.

R.Rajugan, Gardner, W, Chang, E & Dillon, TS 2005b, 'xWeb: An XML View Based Web Engineering Methodology', *International Workshop on Ubiquitous Web Systems and Intelligence (UWSI '05), Colocated with ICCSA '05*, Singapore, pp. 1125-35.

Reynolds, D, Shabajee, P & Cayzer, S 2004, 'Semantic Information Portals', *Proceedings of the 13th International World Wide Web Conference (WWW '04)*, USA.

Schwabe, D, Rossi, G & Barbosa, SDJ 1996, 'Systematic hypermedia application design with OOHDM', *The 7th ACM conference on Hypertext and Hypermedia*, vol. D. Schwabe, G. Rossi, and S.D.J. Barbosa. "Systematic hypermedia application design with OOHDM", The Seventh ACM conference on Hypertext and Hypermedia. 1996., ACM Press.

Steele, R, Gardner, W, Chandra, W & Dillon, TS 2005a, 'Framework & Prototype for Secure XML-based Electronic Medical Records System', *International Journal of Electronic Healthcare*,

Steele, R, Gardner, W, Chandra, W & Dillon, TS 2005b, 'XML Repository Searcher-Browser Supporting Fine-Grained Access Control', *International Journal of Computers and Application*,

Steele, R, Gardner, W, R.Rajugan & Dillon, TS 2005, 'A Design Methodology for User Access Control (UAC) Middleware', *Proceedings of the IEEE International Conference on e-Technology, e-Commerce and e-Service (EEE '05)*, pp. 385-90.

Tatnall, A 2004, *Web Portals: The New Gateways to Internet Information and Services*, ed. A Tatnall, Idea Group Publishing.

Troyer, OMFD & Leune, CJ 1998, 'WSDM: a User Centered Design Method for Web Sites', *7th International World Wide Web Conference.*

W3C-HTML 1997, *HyperText Markup Language (HTML), Rel 4.01 (http://www.w3.org/MarkUp/)*, The World Wide Web Consortium (W3C).

W3C-OWL 2004, *Web Ontology Language (OWL), (http://www.w3.org/2004/OWL/)*, The World Wide Web Consortium (W3C), viewed March 2005 http://www.w3.org/2004/OWL/.

W3C-RDF 2004, *Resource Description Framework (RDF), (http://www.w3.org/RDF/)*, The World Wide Web Consortium (W3C), viewed March 2000.

W3C-SW 2005, *The Semantic Web (http://www.w3.org/2001/sw/)*, W3C.

W3C-WS 2002, *Web Services Activity, (http://www.w3.org/2002/ws/)*, W3C.

Xiaou, R, Dillon, TS, Chang, E & Feng, L 2001a, 'Mapping Object Relationships into XML Schema', *Proceedings of OOPSLA Workshop on Objects, XML and Databases.*

Xiaou, R, Dillon, TS, Chang, E & Feng, L 2001b, 'Modeling and Transformation of Object-Oriented Conceptual Models into XML Schema', *12th International Conference on Database and Expert Systems Applications (DEXA '01) 2001*, vol. 2113, ed. JL Heinrich C. Mayr, Gerald Quirchmayr, Pavel Vogel, Springer.

Chapter 11

Recapitulation, Future Work and Conclusion

11.1 Introduction

In this chapter, we provide a recapitulation of the research presented in this thesis, discuss the benefits of the research drawing out the main conclusions followed by a discussion of future research directions.

First, we provide a brief overview of the flow of the work in the thesis. This, we hope, will enable the reader to understand the direction of the research presented in this thesis, within the context of the problem being addressed.

During the research activity, based on time constraints and the focus, the research was to have acceptable aims and scope to permit the completion of the research within the given time period. Thus, in the future work section below, we set out how the work of this thesis can be extended further with: discussion on the outstanding issues of this thesis, addressing emerging new research issues and foreseen application of the research results.

11.2 Recapitulation

First we look at the flow of this thesis in the context of the chapter organization and structure. This is followed by a summary of content and findings of each chapter.

An introduction to the status and concepts of data models, XML and view formalisms were provided in the first chapter. This was followed by the past and the present state of view research presented in Chapter 2. Based on the findings of Chapter 2, the research problem was formally defined in Chapter 3. Chapter 4 provided a roadmap: an overview of the proposed solution to the problem defined in Chapter 3. Then Chapter 5 presented the core theory, definitions and formal semantics of the solution. This was followed by chapters 6 and 7, where the modelling, design and semantics associated with the Layered View Model (LVM) were presented using two Object-Oriented (OO) modelling languages, namely XSemantic nets and UML/OCL. Chapter 6 also included a detailed discussion of the proposed XSemantic net extensions, model elements and notional semantics. The next three chapters presented three real-world application scenarios for the LVM, where the LVM views were utilized to provide a conceptual framework for modelling and design of: dimensional data in an XML document warehouse framework (Chapter 8), view specification in the Semantic Web paradigm (Chapter 9) and website and portal design in a view-driven, web engineering framework (Chapter 10). Finally this chapter provides a brief outline of our future research directions and conclusion of this thesis.

Chapter 1 provided a detailed introduction to data models, data modelling and views formalisms as they are imperative to understand the main aims of this research and *conceptualization* of the notion of views. Based on these discussions, the existing problems associated with views, lack of database and querying techniques for XML and emerging new challenges for the web and Semantic Web paradigm were identified. Based on these findings, it is envisaged that a view formalism for the XML and other relatively new paradigms require abstraction at a higher-level, namely conceptual level, than is provided for views by the existing structured data paradigms. Also, as a result of the discussion presented in this chapter, the aims and scope of the thesis were formulated taking into account the strong motivating factors addressed and the given timeframe for the research. Based on the scope, the plan of this thesis was presented.

Chapter 2 presented an in-depth analysis of the research done or being carried out in the area of view formalisms. This included past, present and the merging new view formalisms for various paradigms, from relational to Semantic Web. These view models were grouped into four major categories based on data models for which they are designed for, namely: (i) classical (or relational) views, (ii) Object-Oriented (OO) views, (iii) semi-structured (namely XML) view models and (iv) view models for Semantic Web. The purpose of this chapter was to outline and carefully analyze strengths, weaknesses and shortcomings of the existing research and identify core requirements for the view formalism for XML. The findings included: the lack of high-level modelling and specification formalisms for views, strong dependency of view definitions on query languages, lack of semantics to describe the views outside the query languages and foreseeable problems that may persist if the traditional view notion is adapted to the semi-structured data (e.g. XML) and Semantic Web paradigms.

Given the motivation and scope of the thesis, based on the findings of the persisting problems with the view formalism, the problem definition for this thesis was given in Chapter 3. First, some of the core concepts were defined and described that are imperative to define the problem addressed in this thesis in a precise and comprehensible manner. The problem was defined such that, it is concrete and solvable within the scope and timeframe of the thesis. To provide arguments for the soundness of the problem, some of the feasible solutions were presented and evaluated based on their features, extensibility and soundness. A formal statement of the problem was also given.

Chapter 4 gave an overview of the solution addressed in this thesis. Here, first an overview of the notion of XML views, view concepts, properties and the XML view terminologies were informally presented. This was followed by a brief discussion on modeling and transformation issues related to LVM views which included a discussion on two Object-Oriented (OO) modelling languages, namely XSemantic nets and OMG UML/OCL. Another unique LVM view property, the view hierarchy and how it is dealt with in the LVM, is presented with facts and arguments to counter the existing discussions and arguments in this area. Also included is a brief overview on modelling and designing view-driven conceptual frameworks using the LVM views, an analogy to the OMG's MDA initiatives. This chapter included a detailed description of two example case studies used in this research to illustrate the theory and concepts, namely: (a) a simple conference publishing system (CPS) similar to that of systems such as IEEE Xplore or Springer conference publishing systems; and (b) a real-world e-Logistics system for global cold storage, warehousing and logistics booking system (e-Sol).

The rest of the thesis was divided into three main parts: the semantics, concepts and the definitions of the LVM, modelling and transformation of views in the LVM and case-study and applications of the LVM in real-world scenarios. The first part was addressed in Chapter 5, followed by the second part in Chapters 6 and 7, as a practical walkthrough of the modelling issues and the transformation methodologies associated with LVM views using two different modelling languages. The third part was presented in Chapters 8, 9 and 10.

Chapter 5 presented the theory and concepts associated with the LVM using formal semantics and notations. It included, formal properties, semantics and characteristics of the LVM. To the best of our knowledge, this thesis is one of the first works which provides such an elaborated formal semantics and definition for the notion of views at the conceptual level, in the context of XML. In addition, here we formally defined one of the unique characteristics of the view mechanism: the *conceptual view operators*. This chapter also introduced the transformation methodologies adapted for transforming LVM conceptual semantics to schematic (or logical) and instance (or document) level semantics.

Chapters 6 and 7 provided discussions on view specification, modelling and transformation issues associated with the LVM views and demonstrated by using two OO modelling languages. Both chapters provided an illustrative walkthrough of the unique characteristics of the LVM views using a real-world industrial case study, e-Sol.

Chapter 6 looked at modelling and transformation methodologies for LVM, using an XSemantic net based OO notation. Here, the LVM view design methodology was systemically elaborated upon with a detailed discussion on XSemantic net model elements and its various modelling notations. In addition, this chapter provided special remarks on the advantages and benefits of using the XSemantic nets for modelling LVM views over other OO languages. Also included was the discussion of the modelling and transformation of conceptual operators and intra-view dynamic view properties (such as *generic methods*) to XML query expressions. One of the findings of this chapter is the realization of the extent to which the XSemantic net notations and representations correspond to XML structure, properties and constraints.

Chapter 7 presented another modelling and transformation methodology of our LVM views, using an industry standard OO modelling language; the OMG's Unified Modeling Language (UML). Here, special attention is given to the Object Constraint Language (OCL) to complement UML model elements and the modelling constructs, to explicitly specify view constraints, as proven by the earlier findings in chapter 6. Another finding of this chapter is the conclusion that UML, by itself, lacks the ability to visually describe many semi-structured data semantics and constraints in detail.

The subsequent three chapters formed the third part of this thesis. They looked at data modelling issues associated with three different domains, where our LVM views were utilized to provide *architectural frameworks* to solve issues in: (a) XML data warehousing, (b) ontology bases and views and (c) website (and web portal) development. Each chapter provided comprehensive background research and information related to the research in question (i.e. (a) to (c) above) and provided a detailed discussion of how the LVM views are utilized in developing the frameworks in the related domains, with concepts, definitions and formal semantics.

Chapter 8 looked at designing a view-driven XML warehouse model that is intuitive and comprehensive in modelling dimensional data. The XML warehouse presented here is a top-down conceptual model with three layers of abstraction.

Chapter 9 presented the emerging view model requirement in the Semantic Web (SW) paradigm. First a brief discussion of SW and ontology base is presented together with required characteristics of a view model for SW. Then, analogous to the LVM views for XML, a notion of view concept and semantics were given in the context of ontologies. This chapter also included a discussion of transforming and deploying the proposed ontology views in the Materialized Ontology View Extractor (MOVE) system.

Chapter 10 presented an intuitive approach to modelling and deploying websites and web portals using the LVM. Titled extensible Web (or *x*Web) and extensible portal (or *x*Portal), they are *architectural frameworks* with accompanying design methodologies that are capable of modelling websites (and web portal) using the top-down OO approach.

11.3 Future Work Introduction

Few issues raised during our research were considered to be outside the defined scope of this thesis. However, though not elaborated upon in detail within the chapters of the thesis, here we revisit some of these issues as part of our future research direction.

Our future research directions may be considered as two-fold and can be broadly grouped into two categories, namely:

(i) the proposed extension of the Layered View Model (LVM) for XML and some of the outstanding issues; and

(ii) the adaptation of LVM concepts to Semantic Web and related technologies with applications.

These are discussed in detail below.

11.3.1 Future Research Issues in the LVM for XML

In the LVM, some issues deserve further investigation. Here, we provide some discussion of this.

One such issue is the alternate formal approaches for specifying LVM views using approaches such as fuzzy set (Ma 2005). This in theory may solve some of the issues, incomplete information, incomplete structure, etc., that frequently occur in web scale XML repositories, XML databases (Aguilera et al. 2002) and very large scale XML document warehouses. Using such formal methods to specify high-level LVM views for XML may be utilized in various knowledge discovery applications such as, data mining XML documents for meaningful structures and patterns, explanatory semantics (Feng & Dillon 2003) for XML document warehouses, association-rule mining, etc.

Another outstanding issue which was outside the original scope of this thesis was the development of an expert system (Dillon & Tan 1993) based approach to enable automated transformation of view specifications between the different layers of abstraction in the LVM. To this end, the proposed future research will look at enforcing schemata and conceptual operator transformation using expert system rules, thus providing fully automated, user unattended transformation of conceptual views to their receptive logical and document view definitions. Such research would look at:

(i) providing a Graphical User Interface driven modelling tool, such ArgoUML[1] or IBM Rational Rose[2] to allow one to specify conceptual views using XSemantic nets or UML/OCL, which in turn passed onto the knowledge base (part of the expert system) to enable view transformation and constructing at a high-level model.

(ii) providing optimization schemes for refining and optimizing the document view query expressions that are transformed from the conceptual operators. Such optimization schemes offer the user to the opportunity to select suitable mechanisms for the document view queries to suit the environment in which they are constructed and/or modified. For example a mechanism selected for an XML Document Warehouse (XDW) environment may not satisfy the requirements or be efficient in constructing a view from a distributed XML repository. Also, such optimization mechanisms may also be defined using pre-defined rules in the expert system rule engine.

[1] ArgoUML, http://argouml.tigris.org/
[2] IBM™ Rational Rose, http://www-306.ibm.com/software/rational/

Finally, there is the issue of intra-object dynamic view properties that were not fully investigated for the views in the LVM. Though, at the current state, views in the LVM are capable of representing view behaviour (analogous to method and message specification in classes), as part of the research, only a subset of the findings were elaborated upon (Chapter 6) and published (in (R.Rajugan et al. 2006)) in regards to modelling behavioural aspects of the LVM views. However, further research is required to address some of the outstanding issues, such as formal semantics, representation, transformation and deployment of this aspect is not yet fully explored. One of the intuitive approaches considered for future *dynamic* XML views in LVM is to employ a *frame* like structure (Dillon & Tan 1993), which is used in Artificial Intelligent and knowledge base systems. A frame is a prototypical structure that contains the properties of an object, items or events. The behaviour of a frame is defined using a notion call *daemon* (Dillon & Tan 1993), which is similar to a trigger mechanism in the relational or Object-Oriented system. Thus, allowing such provisions to represent conceptual views in the LVM model may provide explicit structural and behavioural specifications, which are mapped to other levels of abstraction with ease. In the next section, we will look at future research directions in the Semantic Web paradigm that is relevant to the research presented in this thesis.

11.3.2 Future Research in the Semantic Web Paradigm and its Applications

As described in Chapter 9, Semantic Web and associated technologies are relatively new and beginning to gain momentum as the next web platform for industrial and commercial web applications. However, due to the nature of things in SW, it is different from traditional databases and web concepts and requires re-visiting and detailed investigation into providing database like features to SW. This includes ontology engineering, ontology representation and transformation, semantic queries, ontology mining, etc. One such issue that is relevant to the current research requires further research is the notion of views for the Semantic Web Paradigm. Here, we consider some of the further research directions (but are not limited to) and concerns given below.

(i) To extend our notion of LVM ontology views presented in Chapter 9, further investigation and formalization of the notion of ontology views. The proposed notion will include support for flexibility and approximation, for example using fuzzy set approach (Ceravolo & Damiani 2004; Ma 2005).

(ii) Development of a high-level view design and transformation methodologies, analogous to the methodologies developed for LVM for XML, for modelling and designing ontology views.

(iii) Investigation into the notion of 2-Es, that is: elaboration and extension transformations in ontology views and feasible solutions to enforce such concepts. As part of this objective, a proposal to include high-level, language independent conceptual ontology operators to perform the above 2-Es. Here, proposed conceptual operators proposed in this thesis will form the basis for such operators.

(iv) Extension to the modelling languages, specifically for XSemantic net to model ontology views that includes:

a. visual modelling of ontology views;

b. visual modelling of inter and intra structure of view objects; and

c. development of a visual GUI driven XSemantic net modelling tool and model transformation plug-ins, such as schemata transformation of XSemantic nets (Feng, Chang & Dillon 2002) to RDF/RDF-S or OWL schemas.

(v) Investigation into transformation of conceptual (ontology) operators to extended W3C query standards and performance driven algorithms.

(vi) Development of a deployment platform for the proposed ontology views to support:

a. *Extraction* of sub-ontologies analogous to the MOVE system, as described in Chapter 9, but extend to support web user interface driven environment

b. Formalism and methods to provide *Elaboration* to the extracted sub-ontologies

c. Formalism and methods to provide *Extension* for ontology views

d. Guaranteed query performance and Quality of (sub) Ontology (QoO) developed.

(vii) Investigation and possible applications of ontology views in the domains of:

a. Security and Trust for SW (Ardagna et al. 2005; Ceravolo et al. 2006; Ceravolo, Damiani & Viviani 2005; Chang, Dillon & Hussain 2006): By introducing a trust layer that allows one to refine the extracted knowledge according to the user's behaviour;

b. Health Information Systems, where ontology views allow Just-In-time classification and analysis of diseases, injury and medication classification systems; and

c. Bio-medical applications where medical information can be represented in ontology bases and accessed via views.

(viii) A visual semantic query building tool to query the resulting sub-ontologies and perform further view definitions and or semantic aware querying.

11.4 Conclusion

The LVM model presented in this thesis provides the solid foundation for conceptualizing the notion of views. We also used XML as a case in point and provided explicit semantics to specify and define LVM views. Moreover, due to its syntax independent high-level (conceptual) view definition, it is easily extendable to the new and emerging data paradigms. It is also envisaged that, in addition to the applications, further utilization of developing conceptual frameworks in the context of Sematic Web applications are also feasible. We hope that the LVM and the associated semantics presented in this thesis are of use to many researchers and future application frameworks will be using this as the basis for further framework developments in XML and Semantic Web paradigms.

References

Aguilera, V, Cluet, S, Milo, T, Veltri, P & Vodislav, D 2002, 'Views in a Large-Scale XML Repository', *The VLDB Journal — The International Journal on Very Large Data Bases*, vol. 11(3), no. 3, pp. 238-55

Ardagna, CA, Damiani, E, Vimercati, SDCd, Fugazza, C & Samarati, P 2005, 'Offline Expansion of XACML Policies Based on P3P Metadata', *Proceedings of the ICWE*, Springer, Sydney, Australia, pp. 363-74.

Ceravolo, P & Damiani, E 2004, 'Fuzzy Mining Class Hierarchies from XML-based Authentication Data', *Proceedings of FuzzyDays04, Computational Intelligence. Theory and Applications*, Springer.

Ceravolo, P, Damiani, E, Elia, G & Viviani, M 2006, 'Bottom-up Extraction and Main-tenance of Ontology-based Metadata', in *Fuzzy Logic and the Semantic Web*, Elsevier.

Ceravolo, P, Damiani, E & Viviani, M 2005, 'Adding a Peer-to-Peer Trust Layer to Metadata Generators', *Proceedings of the OTM '05 Workshops*, pp. 809-15.

Chang, E, Dillon, TS & Hussain, FK 2006, *Trust and Reputation for Service Oriented Environments: Technologies for Building Business Intelligence and Consumer Confidence*, John Wiley and Sons, UK.

Dillon, TS & Tan, PL 1993, *Object-Oriented Conceptual Modeling*, Prentice Hall, Australia.

Feng, L, Chang, E & Dillon, TS 2002, 'A Semantic Network-based Design Methodology for XML Documents', *ACM Transactions on Information Systems (TOIS)*, vol. 20, No 4, no. 4, pp. 390 - 421

Feng, L & Dillon, TS 2003, 'Using Fuzzy Linguistic Representations to Provide Explanatory Semantics for Data Warehouses', *IEEE Transactions on Knowledge and Data Engineering (TOKDE)*, vol. 15, No. 1, no. 1, pp. 86-102

Ma, Z 2005, *Fuzzy database modeling with XML*, Kluwer international series on Advances in Database Systems, Springer, NY, USA.

R.Rajugan, Chang, E, Feng, L & Dillon, TS 2006, 'Modeling Dynamic Properties in the Layered View Model for XML Using XSemantic Nets', *International Workshop on XML Research and Applications (XRA '06) to be held in conjunction with APWeb '06*, Springer-Verlag, Harbin, China, pp. 142-7.

Glossary

Context: The term context refers to the portion of a domain that interests an organization as a whole. It is usually denoted as ζ_{ontext} in this research.

Data Model: A representation that describes data in a given domain that is computationally meaningful. Given a data model, it can refer to conceptual, logical or physical (data storage). In this research, unless explicitly stated, a data model does not refer to physical storage structure.

Data Engineering: A set of formalisms and methods to develop a data model that is easy understand, describe and represent at given level of abstraction. This is also known as data modeling. Here we use these terms interchangeably.

Layered View Model: LVM for short, it is a conceptual framework to model, specify and define views with levels of abstraction.

Object Constraint Language (OCL): A declarative language based on mathematical theorems to specify constraints associated with OO conceptual model elements that represented using UML.

Unified Modeling Language (UML): A standardized and well-known graphical notation proposed the Object Management Group (OMG) to enforce modeling of software system using the Object-Oriented approach. It is based on Meta Object Facility (MOF) language model.

View: Informally the term view refers to a certain perspective of the *context* that makes sense to one or more stakeholders of the organization or an organization unit at a given point in time

W3C: The World Wide Web Consortium, http://www.w3c.org

XML Data Model: The representation of the XML document structure and its semantics. Here, it does not refer to the physical (storage) structure. It is also referred to as XML Document Model.

XML document: A document with Extensible Markup Language (XML) markups and is a simple text format that was derived from SGML (ISO 8879). XML standard was proposed and is maintained by World Wide Web consortium and it is in its third edition of version 1. In this research, unless explicitly stated, the terms XML data (data-centric documents) and XML documents (document-centric documents) are used interchangeably.

XML Schema: XML Schema is a schema definition language, proposal by World Wide Web consortium to describe, constrain and validate XML document instances. Its predecessor id Document Type Definition (or DTD, for short)

XQuery: A native query language recommendation by the W3C for querying XML documents.

XSemantic nets: A graphical notation to model and represent XML domains with OO conceptual level semantics in great detail.

Appendix A

Selected Publications
