Web extensions to UML: Using the MVC Triad

B. Henderson-Sellers, D. Lowe and A. Gu
University of Technology, Sydney

1 INTRODUCTION

Current modelling languages for web system development were examined by Gu *et al.* (2002) against a set of developed criteria or requirements for a Web Modelling Language (WML). Of the six current WMLs examined, some with hypermedia roots, others extensions to OO modelling languages such as the UML (OMG, 2001), all were found deficient in part. This is because those WMLs with a hypermedia basis are more closely focussed on the information architecture whereas modelling-derived WMLs are more closely focussed on the functional architecture. The spread of WMLs along these two important axes is shown in Figure 1, where it is seen that there are no candidates which address *both* the informational and functional architectures concurrently. This paper attempts to move WMLs towards this "target zone" (Figure 1).

Firstly, the scope and objectives of the extension are discussed (Section 2), followed by the proposed model extensions (Section 3) and UML diagram extensions (Section 4). These extensions are then discussed in terms of how they might be used in different design components of the overall software development process (Section 5). Then in Section 6, we illustrate the feasibility of this approach in terms of a small case study.

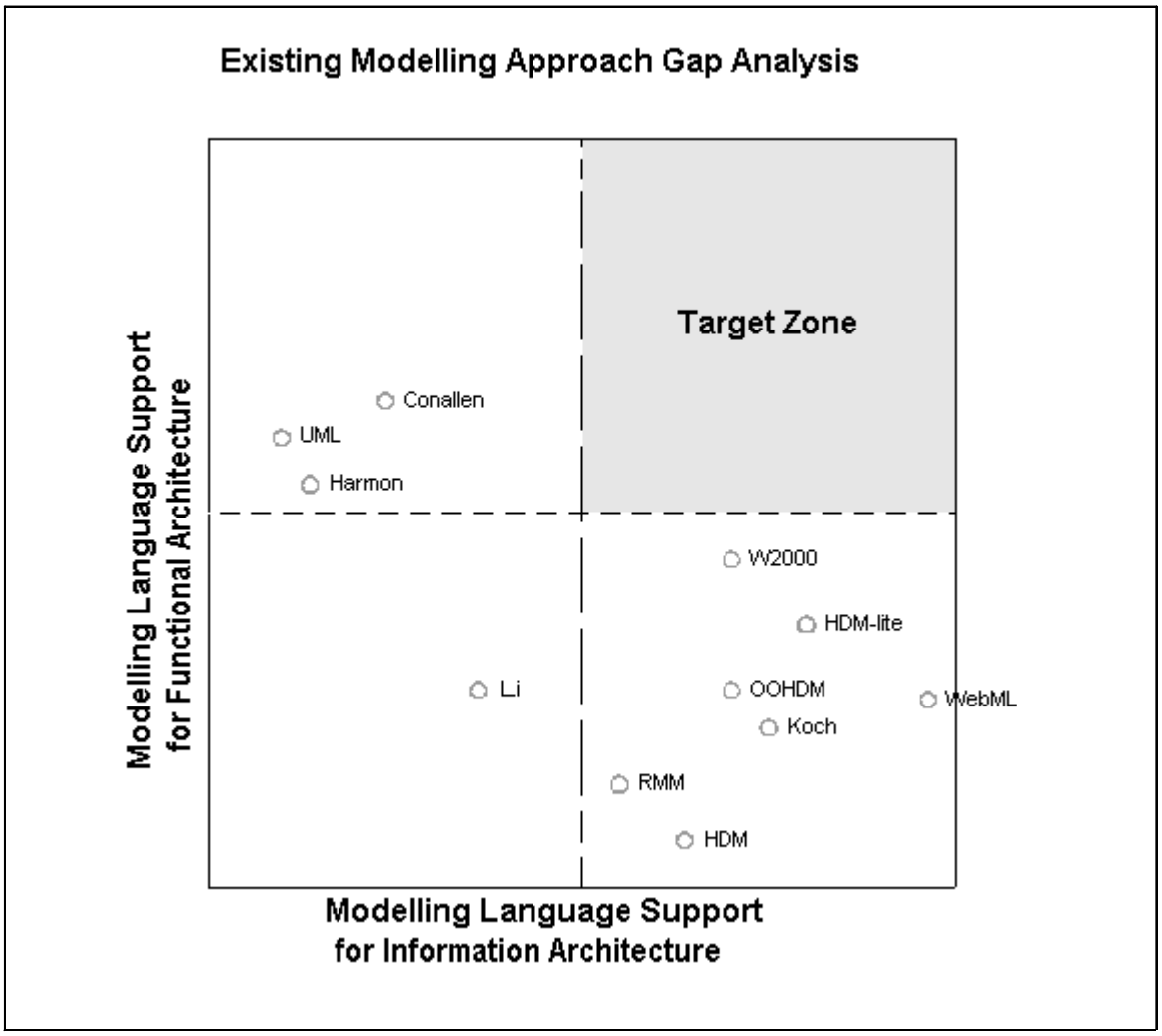

Figure 1: Existing Modelling Approach Gap Analysis (after Gu *et al.*, 2002)

2 SCOPE , OBJECTIVES AND APPROACH

As discussed in Gu *et al.* (2002), limitations and flaws can be identified in the existing modelling approaches that are currently used to develop Web systems. Some of the gaps, such as the inability to support system life cycle management and the potential misuses of UML extension mechanisms, need to be addressed in separate research projects and are beyond the scope of this present paper. Here we address the issues relating to the need to model increasingly sophisticated functionality and, most

importantly, to integrate the functional architecture with the informational architecture.

Since the extensions proposed in this report make use of UML extension mechanisms, this approach needs to be defined explicitly in order to avoid confusion. The normal way to extend the UML is the use of stereotypes. "A stereotype is not the same as a parent class in a parent/child generalization relationship. Rather, you can think of a stereotype as a metatype, because each one creates the equivalent of a new class in the UML's metamodel." (Booch *et al.,* 1999, p80). In other words, stereotypes extend the UML (M_2) metamodel indirectly at the M_1 (or model) level, wherein the relationship between a class and its stereotype is an "instance_of" rather than an "is_a_kind_of" relationship (Atkinson *et al.,* 2000). Whilst this extension mechanism makes it easy for users to extend the UML notation as and when they want to, it may introduce confusion and semantic problems because the (mis)use of the inheritance relationship in a stereotype does not truly reflect the intended instantiation relationship that is used in direct metamodelling (Atkinson, 1998; Atkinson *et al.,* 2000; Atkinson & Kühne, 2000, 2001).

However, despite the inherent problems of the current UML stereotype concept, for the purpose of this project and this paper, we will still use stereotypes as the extension mechanism, mostly due to the support given to them by CASE tools. To further improve the proposed extensions, direct modification to the UML metamodel may need to be considered as an alternative (see e.g. Henderson-Sellers *et al.*, 1999).

The objectives of the extensions are:

- To address the deficiencies identified in the the gap analysis reported in Gu *et al.* (2002). These issues include the inability to model sophisticated functionality, the disconnection between the information architecture and the functional architecture, the disconnection between the business model and the technical architecture and the inability to support modelling at various abstraction levels.
- To ensure the integrity of the resultant Web system architecture, whilst merging the business model with the functional and information aspects of the technical architecture and representing system structure at different abstraction levels.

Based on the analysis of the existing modelling approaches, we propose a UML extension with information modelling concepts taken from other modelling approaches, WebML in particular (Ceri *et al.,* 2000). This option is chosen because:

- The UML notation is commonly used and accepted. It appears to provide reasonable support for system functional architecture modelling. To provide sufficient support for Web system development, some Web specific features need to be defined; and
- Approaches from a hypermedia background demonstrate reasonably rich and balanced support for the information architecture and concepts from them can be used as the foundation for the extension. Amongst them, WebML is a more recent attempt that provides modelling capabilities for most critical aspects of Web system information architecture.

3 EXTENSION MODEL STRUCTURE

To ensure the architectural integrity of Web systems, we would like to propose an extension model structure that can be used to support the modelling of the information

architecture and the functional architecture in a coordinated and cohesive manner. This structure is based on the MVC concept.

3.1 THE MVC CONCEPT

"The Model-View-Controller architecture, often known just by the letters MVC, has been a feature of Smalltalk since Smalltalk-80. It is based on the concept of separating out an application from its user interface" (Hunt, 1997, pp266). The responsibilities of Model, View and Controller are described as follows:

- Model – the information model that handles data storage and information processing. It manages the behaviour of the data in the application domain.
- View – handles how the information is displayed visually, which is the interface part of the system.
- Controller – provides user interaction to, or control of, the information models.

Some well-known reasons for its popularity are (Hunt, 1997):

- Reusability of application and / or user interface components;
- Ability to develop the application and user interface separately; and
- Ability to inherit from different parts of the class hierarchy.

Initially used in object-oriented programming languages, such as Smalltalk, as a code level structure, the MVC concept has gained more recognition in recent years and has been applied at the design level, such as in the design patterns in J2EE (Sun, 2001).

It should be noted that MVC is typically used as a specific architecture rather than a broader modelling framework and, as such, there may be concerns over the extent to

which this limits its applicability to modelling a broader range of applications and systems. This issue is recognized but not addressed further in this paper.

A thorough study of the existing modelling approaches, especially the ones that support the information architecture reasonably well, such as OOHDM (e.g. Schwabe and Rossi, 1998) and WebML (Ceri *et al.*, 2000), demonstrates that the separation of modelling entities in the conceptual model and the interface entities in the presentational and navigational models has been used to provide advantages to these well-established approaches, which include:

- The better understanding of system architectural issues brought by the separation of concerns; and
- The possibility of both flexibility and personalization provided by the match of the conceptual model with different presentational or navigational models.

Whilst we believe that these approaches can provide reasonable support for information architecture modelling, the functional architecture aspect is normally weak or even absent in these approaches. We therefore propose the extension model structure – a modified MVC architecture as shown in Figure 2 – which is explained in detail in the following three subsections.

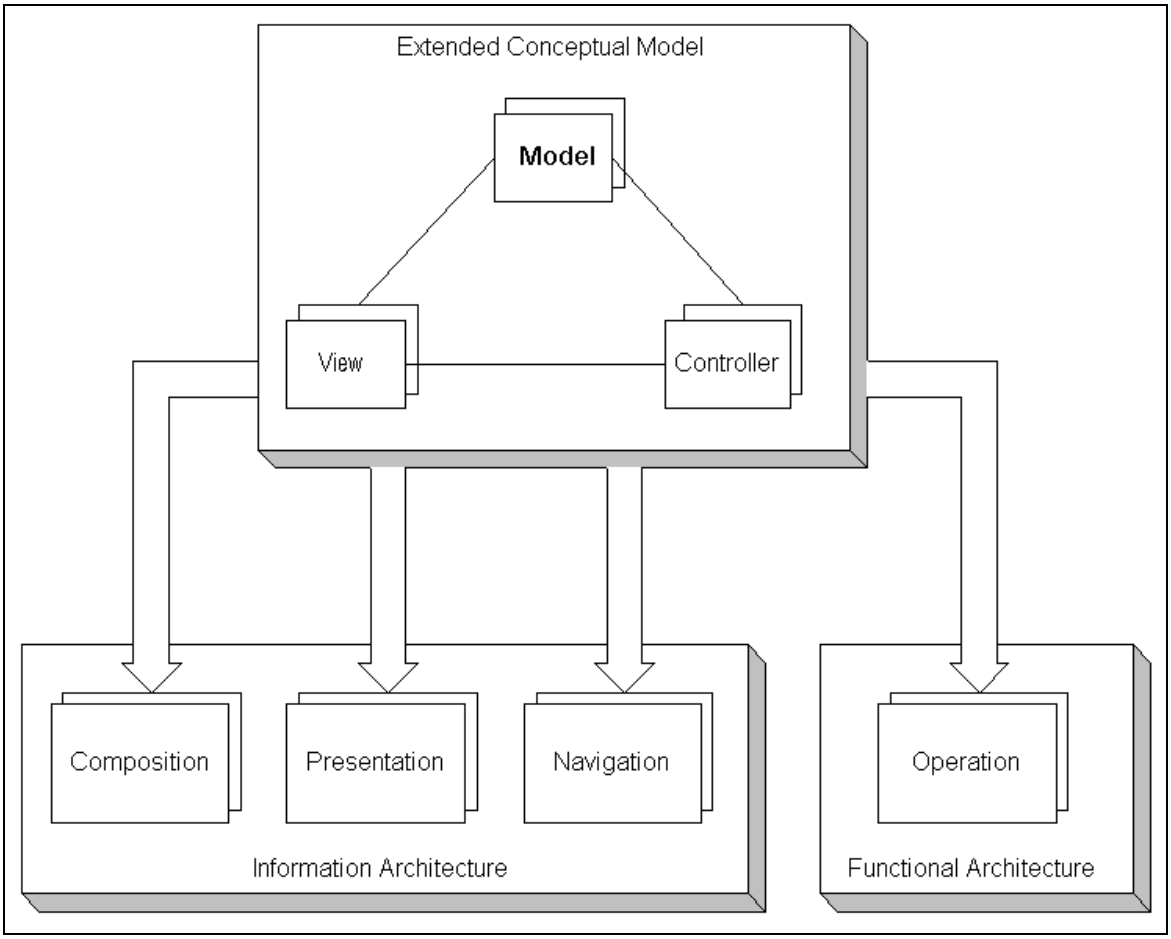

Figure 2: Extension Model Structure

3.2 EXTENDED CONCEPTUAL MODEL

The extended conceptual model contains three types of elements:

- Model – the normal model element that represents business entities;
- View – defines the composition of Models at the interface level; and
- Controller – defines object behaviours. These behaviours can occur either at the interface level, which are navigational behaviours, or at the back-end, which are system functions.

For example, business or application domain entities, such as "student" and "product", are Models in this extended conceptual model. At the interface layer, though, these entities can be displayed in various forms. When "student" and

"product" are displayed in the form of list, they can be View "List"; they can also be displayed as View "DataUnit" if detailed information is required. An example of Controller is the index used to sort the "student" or "product" List Views. If the "product" List needs to be displayed by category, then an "Index" Controller "index by category" can be connected to the View to fulfil this requirement.

There are multiple Models, Views and Controllers in the extended conceptual model. Whilst each Model represents a business entity, the Views and Controllers, which are connected to the Models, represent and specify the interfaces and behaviours of the Models. This extended conceptual model structure has several consequences:

- In order to represent the entities of the business domain and to link these entities to the logical concepts in the technical architecture that defines and implements the interfaces and behaviours of the business entities, Models, Views and Controllers need to be used together and their interconnections need to be carefully defined and managed.
- By connecting multiple Views and Controllers to the same Models, various versions of the interfaces and behaviours of the business entities can be defined. This structure can be used to support personalization on the one hand, and the flexibility and dynamicity of the system architecture on the other.
- The Model-View-Controller concept provides the capability to separate architectural concerns into various aspects and abstraction levels and, in turn, helps to manage the complexity of Web systems.

The design of Web system information architecture and functional architecture is performed on the foundation of the extended conceptual model.

3.3 INFORMATION ARCHITECTURE

Various aspects of the information architecture are modelled using different modelling artefacts. These aspects include:

- Composition: Defines the model structure in terms of Model, View and Controller.
- Presentation: Defines the interface level concepts. Presentation is based on the Views in the composition model.
- Navigation: Defines the navigational structure and behaviour. Navigation is performed by Controllers. When activated, Controllers pass control to one another and the application therefore flows from one part of the system to another. The result of navigation is the change of Views.

3.4 FUNCTIONAL ARCHITECTURE

Operations need to be modelled in the functional architecture by:

- Using existing UML concepts and diagrams, such as the statechart diagram and the sequence diagram.
- Using View and Controller defined in the extended conceptual model. For example, if when activated, instead of passing control to another Controller in the same or another View, the Controller passes control to a Controller that performs back-end functionality, then an operation is invoked. The end result of an operation can be a state change in the system, either at the user interface level or at the back-end level.

4 UML DIAGRAM EXTENSIONS

Several additional diagrams need to be defined in the UML notation to support the proposed model structure. These diagrams are shown in Figure 3 and described in Table 1 in which three new stereotypes of Classifier are introduced: «view», «controller» and «presentation». It is worth noting that the UML diagrams and their interconnections presented in Figure 3 only demonstrate one possibility of using the proposed UML extensions to support Web system development. Therefore, this diagram is to illustrate one option, rather than a complete prescription for defining a process or framework. This is due to:

- The existing UML notation provides possibilities for users to use only a subset of its diagrams according to their particular modelling requirements; and
- The extension proposed in this paper aims to suggest research directions and practical application, while maintaining the flexibility of the UML notation.

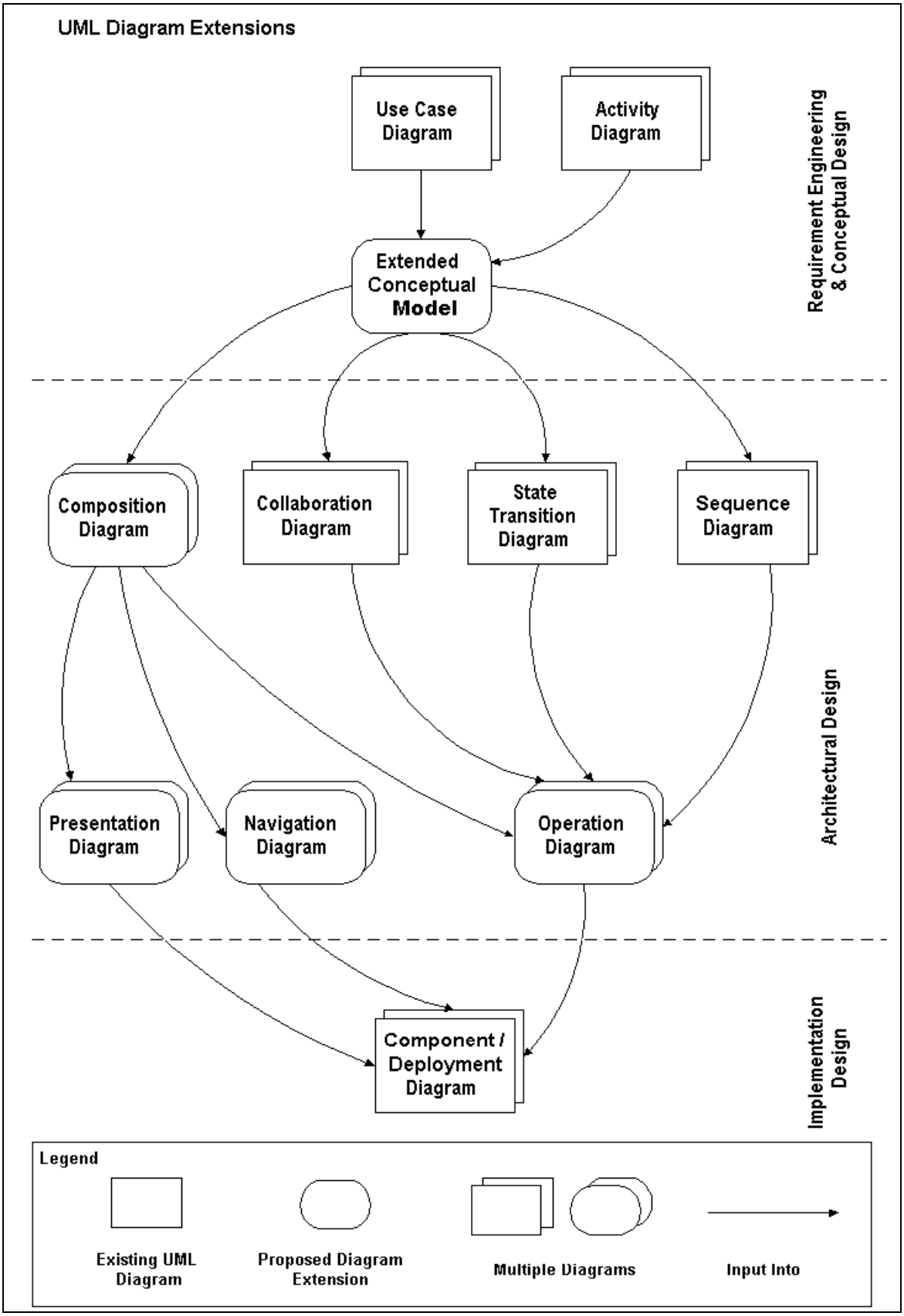

Figure 3: UML Diagram Extensions

Table 1: UML Diagram Extensions

Diagram	Extension Needs	Extensions
Use Case Diagram	N/A	N/A
Conceptual Model	This model uses those concepts from the problem domain that are independent of the technologies used to build the system. The proposed concept of extended conceptual model links these basic conceptual entities (Models) to Views and Controllers.	Although the concept of extended conceptual model is introduced, it does not necessarily mean that new modelling artefacts need to be defined in the conceptual model; instead, it only implies that the conceptual model contains Models and their corresponding Views and Controllers, from a semantical perspective. The relationship between Model, View and Controller can be represented in the composition diagrams, which will be discussed later in the table.
Activity Diagram	N/A	N/A
Collaboration Diagram	N/A	N/A
Statechart Diagram	N/A	N/A
Sequence Diagram	N/A	N/A
Composition Diagram	This diagram does not exist in the UML. Coming from WebML, it covers one aspect of information architecture of Web	Stereotypes «view» and «controller» are defined on the UML class diagram. «view» is defined in order to show different components of the interface at a

	applications.	relatively high level. «view»s can contain other «view»s, in which case it makes up a view – sub-view hierarchy. «view» or sub-«view» can be a Web page, part of a Web page, or a combination of several Web pages. «view»s can contain not only other «view»s, but also «controller»s. «controller»s perform functions – navigations or operations, either on the interface, or behind the screen. The «controller»s can be further defined as different classes e.g. Index, Filter, DataUnit, and Operation. Controller can navigate from one «view» to another, either in a contextual or non-contextual manner. When a «controller» initiates a function from the interface, it can then activate other «controller»s, either on the client side or on the server side. At the end of a function, control can be passed back to a «controller» on the interface, i.e., a DataUnit «controller», so that the user can interact with the system again.
Presentation Diagram	This diagram does not exist in UML and needs to be defined to support the representation of interface-level modelling, such as the components	«presentation» elements, such as Page, Filter, DataUnit and Button, are defined to show screen mock-ups. These elements specify the presentational aspects of «view»s. «view»s are at a higher abstraction level than

	that make up a Web page.	«presentation» elements, and represent a selection of data from the extended conceptual model. Instead of using the existing UML diagrams, the presentation diagram is defined as a new type of diagram. This is largely due to the lack of support for presentation level modelling in the existing UML notation. Modification and personalization can be done by matching different «presentation» elements to the same «view»s. Ideally, presentation diagrams should be generated automatically by a CASE tool, according to the semantics in the composition diagrams. Users can modify them manually if required. Style is defined as a class with the stereotype «presentation» to show the style or format on the interface level. It can be generic and linked to, and indeed reused by, many «presentation» elements; or it can also be specific and used to define particular presentational characteristics of some individual «presentation» elements. By defining a number of different «presentation»s for each «view» and defining and linking a number of S«tyles to each «presentation», flexibility and personalization can be achieved in Web

		system design.
Navigation Diagram	This diagram does not exist in the UML and needs to be defined.	At a high level, navigations are performed by moving from one «view» to another. For each user or user group, navigation diagram(s) can be defined to show to which «view»s the user has access and how they are interconnected. At a lower level, navigations are performed through «controller»s. When a user invokes a «controller», the display changes, either by going to another «view» (i.e., another page) or to another part of the «view» (i.e., goes to the Top). The «controller»s can carry contextual information. When a «controller» is invoked to perform an operation, the user "loses" control of the system. The «controller» either performs a function itself or consequently activates another «controller» to complete the function. Once the function is finished, control can be passed back to an interface level «controller» and the user regains the control over the system.
Operation Diagram	Operations are currently modelled in the UML using diagrams such as the collaboration diagram and the statechart diagram. However, in the proposed model structure,	Simple operations can be represented in detailed level navigation diagrams, whilst complex operations may need to be defined in further detail using operation diagrams. In an operation diagram, the flow of

	operations also need to be represented using View and Controller. This is to ensure that the functional architecture is modelled using the same concepts as in the information architecture, so that the two aspects can be connected in a reliable and consistent fashion.	operations is represented by the passing of control amongst «controller»s. These «controller»s can reside either on the client side or on the server side.
Component Diagram	N/A	N/A
Deployment Diagram	N/A	N/A

5 USING THE EXTENDED UML DIAGRAMS

In this section, the extended model structure (shown in Figure 3) and UML diagram extensions (described in Table 1) will be studied using a partial development process. This does not, in any way, imply that the proposed extensions need to be used in conjunction with any particular process; rather, the process described here is used as an example to demonstrate the usage of the extensions and diagrams during the course of Web system development.

5.1 REQUIREMENTS ENGINEERING & CONCEPTUAL DESIGN

During the requirements engineering and conceptual design stages, business requirements are captured. As a result, both the business model and the business

processes together with the desired system functionality are modelled and documented in the conceptual model.

5.1.1 REQUIREMENTS ENGINEERING

Business requirements are normally captured using UML use cases and activity diagrams. Since these diagrams represent concepts, their relationships and business processes, which are mainly related to the business domain, no extension needs to be defined.

During the requirements engineering stage of Web system development, many use case diagrams and activity diagrams can be created, depending on the size and complexity of the system to be developed. Both use case diagrams and activity diagrams are used as input in the creation of the extended conceptual model.

5.1.2 CONCEPTUAL DESIGN

The extended conceptual model is built from understanding the problem domain. The entities in the extended conceptual model are problem domain concepts, not computer system components. The extended conceptual model is the centre of the entire model structure. All other diagrams and modelling elements are based on it and will evolve from it. There can be only one conceptual model of the Web system, although a number of diagrams can be generated to represent the conceptual model at various abstraction levels and from different viewpoints.

To extend the conceptual model, Views and Controllers are then defined based on the original conceptual model. Views define interface composition, at a relatively high abstraction level. Controllers define object behaviours, either at the interface level or in the back-end. Whilst the concept of the Model-View-Controller structure is viewed

as part of the extended conceptual model, the definition is represented in the composition diagrams. To support modelling at various abstraction levels, the composition diagrams consist of two types: composition in-the-large and composition in-the-small. One conceptual model can connect to multiple composition diagrams and can therefore support various definitions on the interface and behaviour levels.

5.2 ARCHITECTURAL DESIGN

Once the extended conceptual model is defined, it can be used as the foundation of Web system architectural design. Other model elements and diagrams can be defined from the extended conceptual model and used to document the system structure at more detailed levels. These design activities may, and often do, occur more or less in parallel. During this stage, some existing UML diagrams, such as the collaboration diagram, the statechart diagram and the sequence diagram can be used to facilitate the modelling process (OMG, 2000).

To better support the modelling of the Web system information architecture and functional architecture, we propose some extension to the UML diagrams and/or modelling elements, which are described in the following subsections.

5.2.1 PRESENTATIONAL DESIGN

To represent presentational design, which is an important integral of Web system information architecture, presentation diagrams are defined from the composition diagrams. They show how views are displayed on the interface level. One view can be matched to more than one presentational definition. Personalization at the interface level is then supported.

5.2.2 NAVIGATIONAL DESIGN

Views and Controllers interconnect with each other, and thus support the navigation in the Web systems. From a navigational perspective, the user navigates between Views through the usage of Controllers. For example, when one button on a web page is pressed, another web page is loaded. Navigation contains two aspects:

- The Static Aspect – Navigational Structure

The definition of interconnections between Views represents the navigational structure of Web systems at a logical level. This navigational structure can be more complicated than a basic linear or tree hierarchy.

- The Dynamic Aspect – Navigational Behaviour

Navigational behaviours in Web systems can typically be contextual or non-contextual. The navigational activities are performed by activating Controllers to pass control, either from one View to another or from one part of a View to another part.

Because of the potential complexity of Web system navigation, the representation of navigational structure and navigational behaviour needs to be supported at different abstraction levels, so that thorough understanding of the navigational aspect can be achieved. This is implemented by using two types of navigation diagrams: navigation in-the-large and navigation in-the-small.

5.2.3 OPERATIONAL DESIGN

Operations are performed by Controllers. This can happen either when a user invokes a Controller by interacting with its related View, or when the system initiates a function by passing control to a Controller.

Some functions need only one Controller to complete, whilst others require the collaboration between several Controllers. In the latter case, control flows from Controller to Controller during the process of the function. When the control is not with the Controller that relates to Views at the interface layer, the user cannot interact with the system.

To perform an operation, other resources such as legacy applications, database files or external links may be required. This is true with navigation as well. A user can navigate from a website to its related links and then return, as part of his/her normal navigational route.

6 CASE STUDY INTRODUCTION

To complete this paper, we will now illustrate the proposed extension model structure and UML notation extensions using a case study. Limited by the sophistication of the case study, some aspects of the proposal may not be demonstrated thoroughly.

The Web page of the case study is shown in Figure 4. This Web system facilitates the online enquiry of the policies, procedures and issues that apply to students who attend courses in the Faculty of Engineering at the University of Technology, Sydney (UTS). For the purpose of this paper, we will call this system UTSE-Guide.

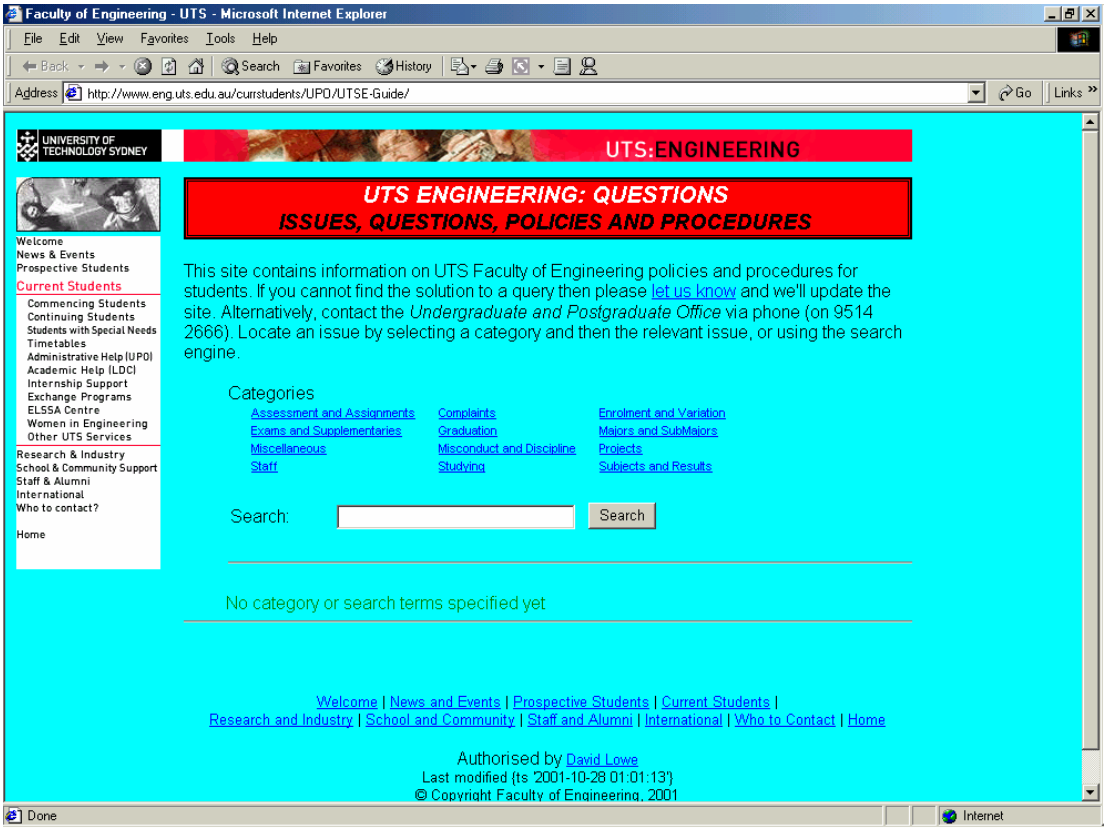

Figure 4: Case Study Web Page

This case study is based on an existing system, which is small in size and for which there was no documentation produced during the development stage. As with normal Web system development, for a system of this size, the extent of modelling as conducted in this case study is not necessary; the modelling process used and the resulting diagrams generated here are purely to illustrate the concepts of the proposed extensions.

6.1 REQUIREMENTS ENGINEERING

The basic requirement of UTSE-Guide is documented in the use case model shown in Figure 5. Users can perform the following functions:

- Search issues by category

A category list will be displayed on screen. To search issues by category, a user needs to select the category name that interests him/her from the list.

- Search issues by keyword

User can input keyword(s) and activate the search based on the keyword(s).

- Display issue list

The result of a search function, be it search by category or search by keyword, is a list of issues that satisfy the search criteria.

- Display issue details

Details of the issues can be displayed when a particular issue is selected from the issue list.

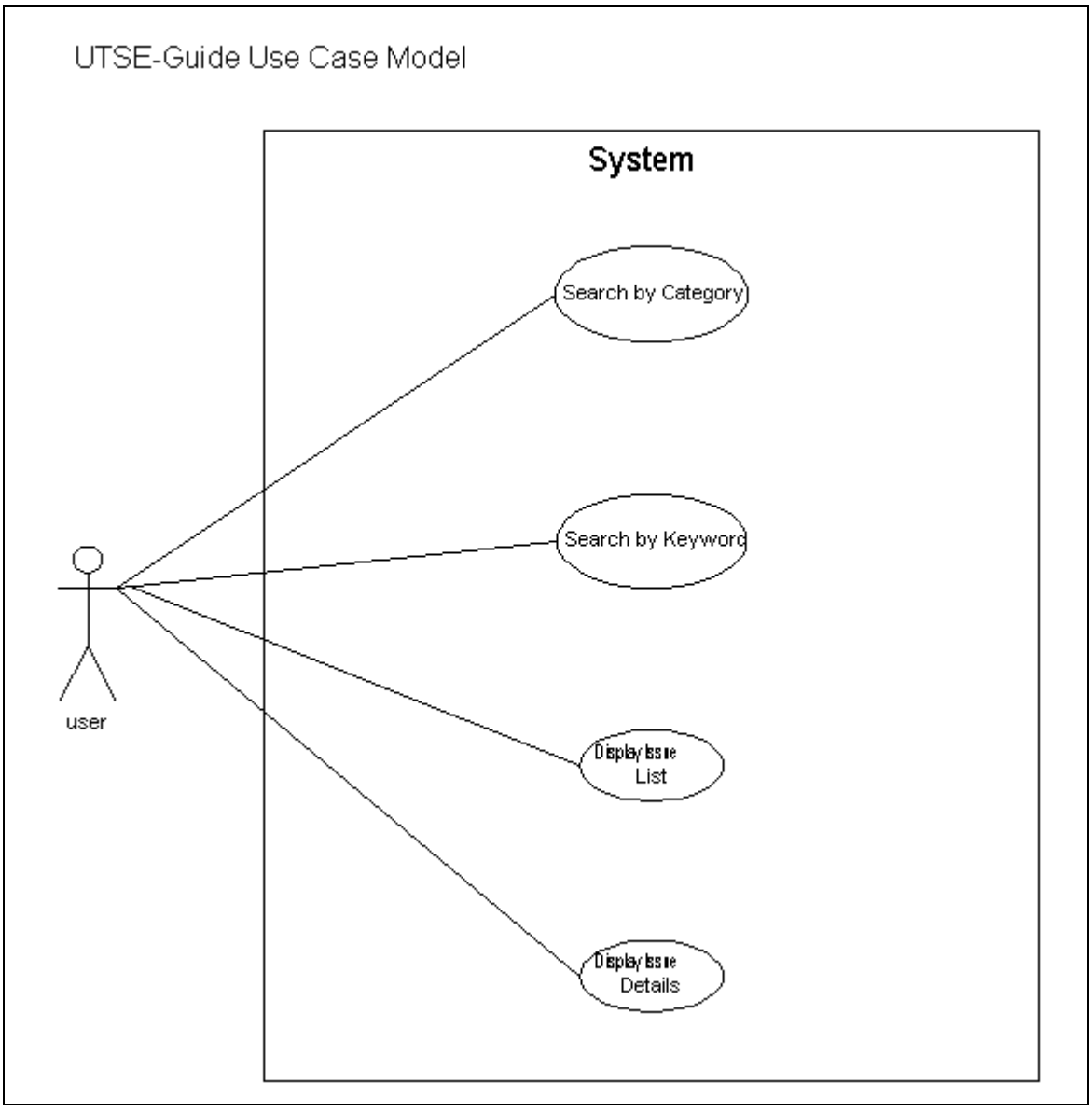

Figure 5: Use-case model for the UTSE-Guide

6.2 CONCEPTUAL MODEL

The conceptual model of UTSE-Guide is shown in Figure 6. The core business entities in UTSE-Guide include:

- Category: The grouping of issues.
- Issue: The definition of problems, their solutions and related procedures and references. An issue belongs to one category.
- Question: Policy or procedure related queries. A question belongs to one issue.
- Keyword: Keywords are related to issues and are used to perform search functions.
- Student: The target audience of UTSE-Guide. Issues can apply to one of many student types, i.e., undergraduate and postgraduate.

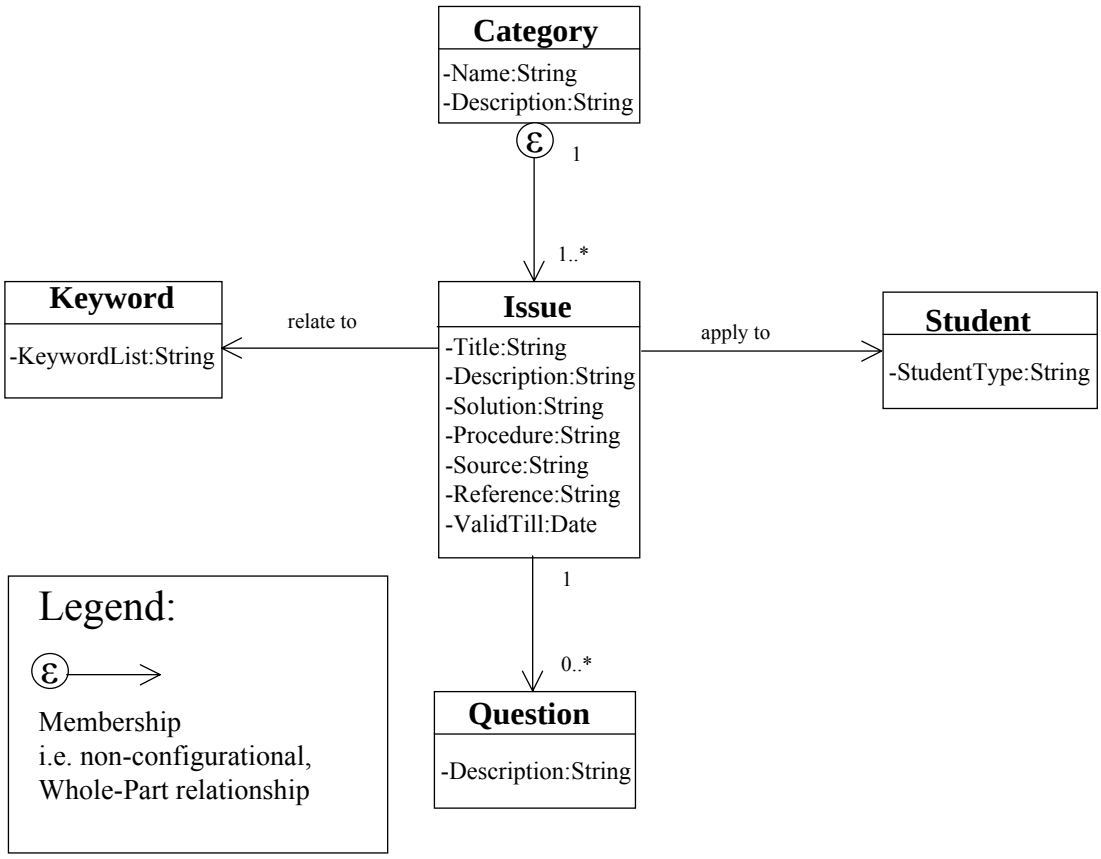

Figure 6: Conceptual Model

The relationships amongst the business entities are:

- Category-Issue: Issues are grouped into various categories. One category can contain zero to many issues. That means, even when a category contains no issue, its definition is still valid in the UTSE-Guide system. When a search is issued by category, all issues in the specified category are then returned as the search result.
- Issue-Question: Questions are grouped to relate to various issues. One issue can contain zero to many questions. The definition of an issue is valid, even when it contains no question.
- Issue-Keyword: Issues are related to keywords. These keywords are used during search functions. When a search is performed by keyword, all issues related to the specified keyword(s) are returned.

- Issue-Student: Certain issues only apply to certain student types. For example, issues related to postgraduate courses only apply to postgraduate students. One issue can apply to one or many student types.

6.3 INFORMATION ARCHITECTURE

The information architecture is represented using composition diagrams, navigation diagrams and presentation diagrams. Both composition and navigation diagrams can be constructed at different abstraction levels.

6.3.1 COMPOSITION IN-THE-LARGE

The high level structure of the UTSE-Guide composition is shown in Figure 7. This high level diagram is called composition in-the-large.

From an information structure perspective, the UTSE-Guide system can be stereotyped as a «view» class, with the two major components also represented as «view»s: Issue Search and Issue Display. There is a strong connection and coincident lifetime of the parts (the sub-«view»s) with the whole (the «view») for which we use (for present purposes) the UML black diamond notation. This high level composition demonstrates the information structure of the system at an abstract level, thus providing the developers with the big picture.

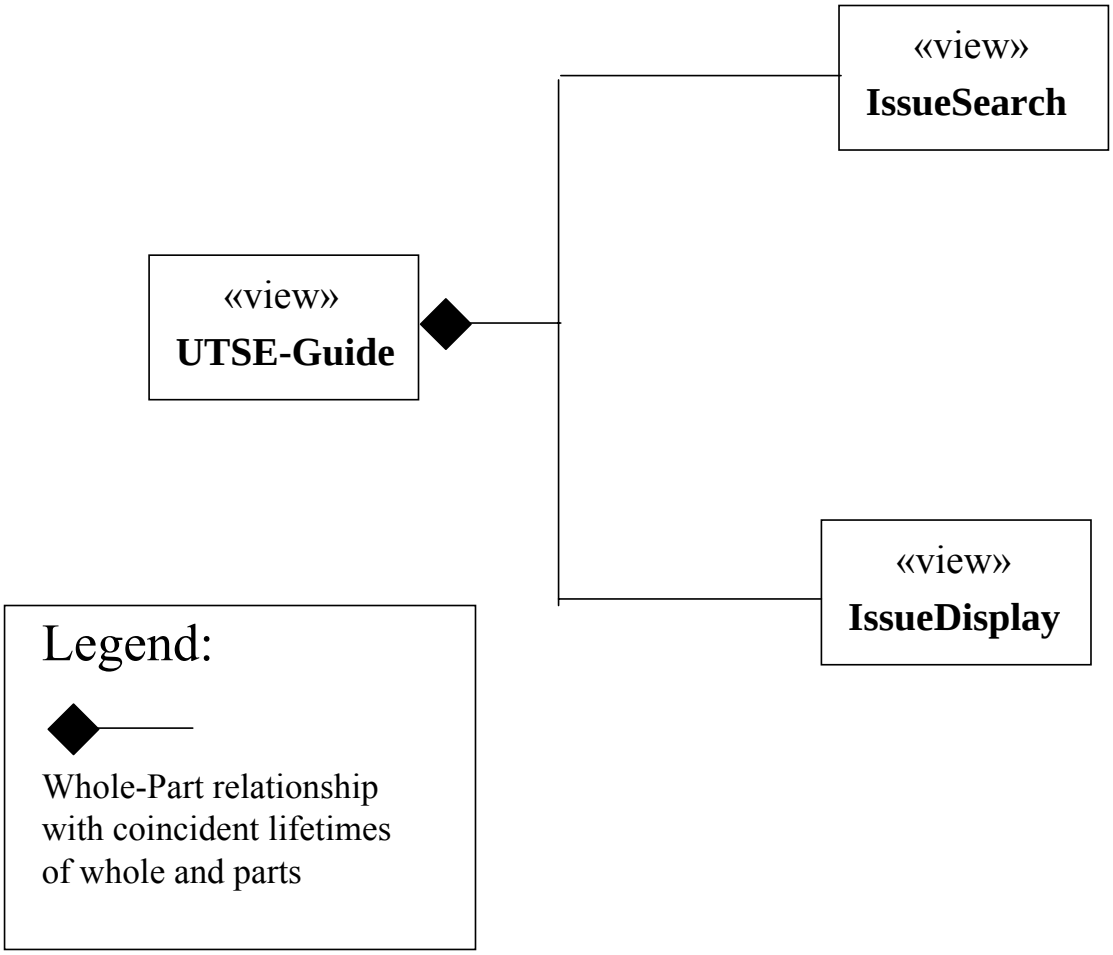

Figure 7: Composition In-The-Large for UTSE-Guide

6.3.2 COMPOSITION IN-THE-SMALL

To further specify system composition at a more detailed level, diagrams can be constructed to document the composition in-the-small (as shown in Figure 8). This diagram documents the composition of the component "Issue Search" at a more detailed level, i.e., Web page level. View "Issue Search" is further broken down into Views with associated Controllers. Some typical types and interconnections of Views and Controllers are explained below:

- Unactionable Views

Some Views are not actionable. In other words, they are purely display fields at the user interface layer. View "Issue Search", which happens to be a Web page in this

case study, contains Title and Text Views that are defined only to serve display purposes. They cannot be activated to trigger any navigation or operation.

Unactionable Views do not link to Controllers.

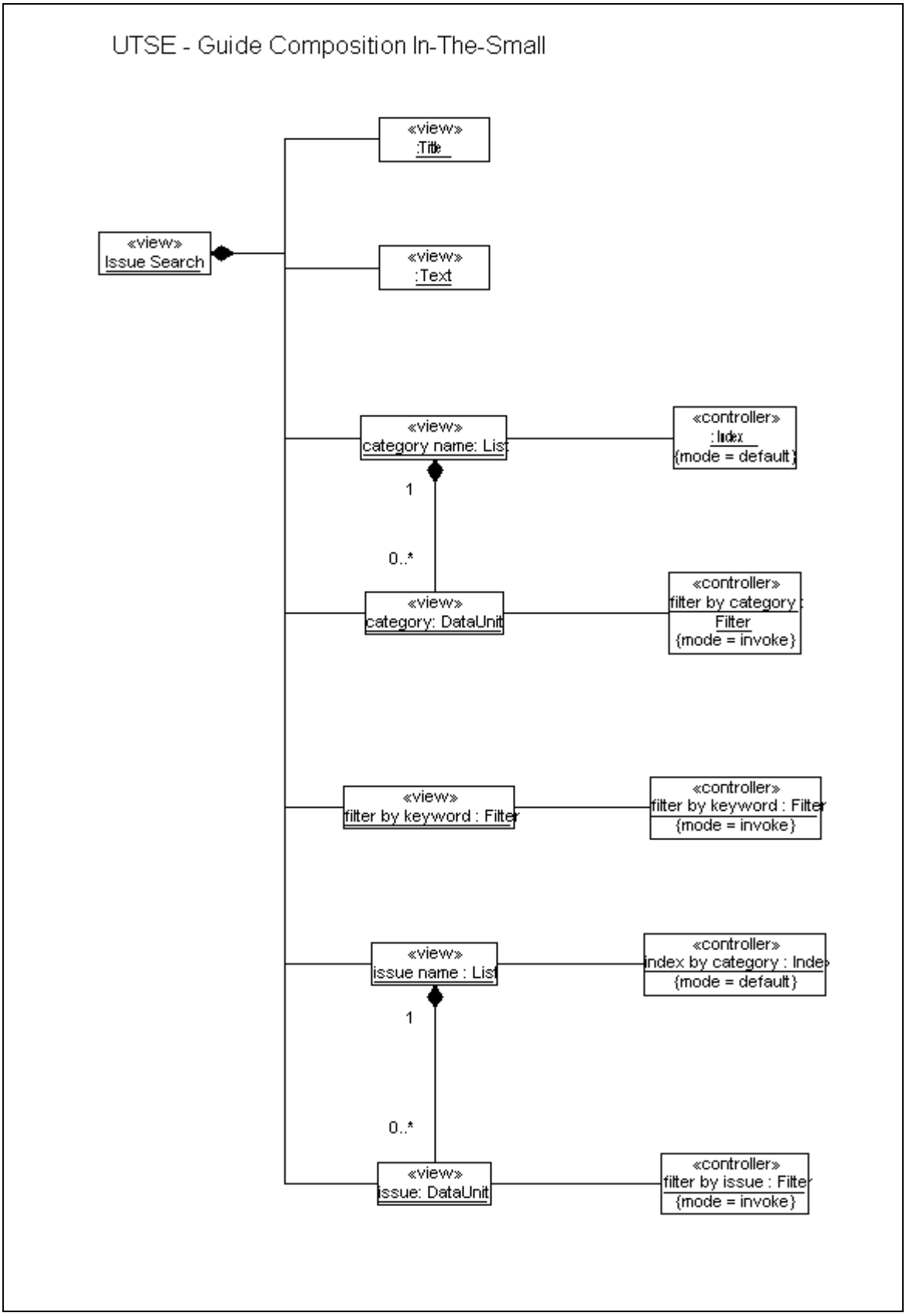

Figure 8: Composition Diagram – in-the-small

- Actionable View and Controller Pair

To make Views “actionable”, they need to be paired up directly with Controllers. When implemented together, a View-Controller pair can provide navigational and/or operational capability.

For example, in UTSE-Guide, «view» “issue name” and «controller» “issue by category” are connected. Whilst «view» “issue name” displays a list of issue names, «controller» “issue by category” provides the indexing function to sort issues by category. Used together, they implement the display of issue names that are sorted by category. «controller» “issue by category” is activated with “mode = default”. This implies that this «controller» is activated whenever the main «view» “issue search” or the sub-«view» “issue name” is displayed or refreshed.

- View and Sub-View

Views can contain sub-Views. For example, «view» “issue name: List” contains multiple «view» “issue: DataUnit”. This means that «view» “issue name” consists of a list of issue names, whilst «view» “issue” consists of a DataUnit “issue”. Data units “show information about a single object, e.g., an instance of an entity or of a component” (Ceri *et al.*, 2000). As demonstrated in the composition diagram, «view» “issue name” connects to multiple «view» “issue”, which implies that the issues listed in the “issue name” «view» relate to more than one “issue” objects, or so-called “DataUnits”.

6.3.3 NAVIGATION IN-THE-LARGE

The high level navigation of the UTSE-Guide system is shown in Figure 9. As discussed earlier, the demonstration of the UML extensions and their proper

applications in Web system development may be restricted by the lack of sophistication in the case study system.

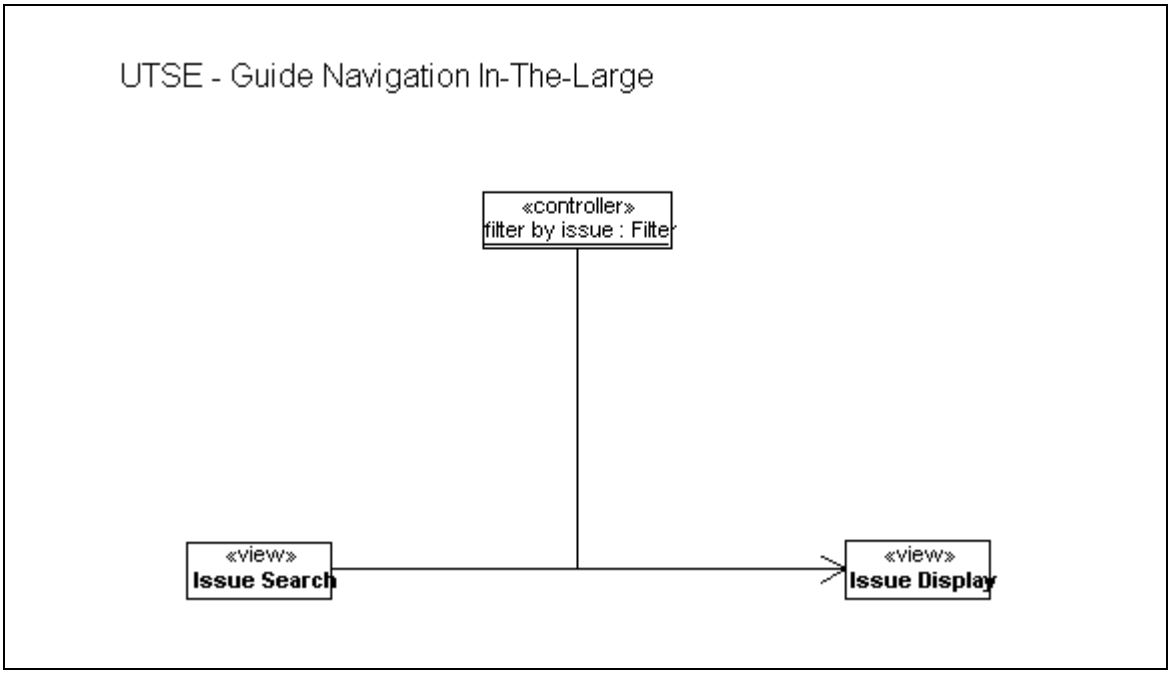

Figure 9: Navigation Diagram – In-The-Large

At a high abstraction level, the navigational structure of the UTSE-Guide system can be represented as the connections of the “Issue Search” «view» and “Issue Display” «view». The navigability between them is shown as an arrow from “Issue Search” «view» to “Issue Display” «view». This is not to say that users cannot return to the “Issue Search” «view» once a particular issue has been displayed in detail; instead, we view the “Return” function as part of the built-in features of the Web browser and it is therefore unnecessary to be specified explicitly in the navigation diagram.

User can navigate from «view» “Issue Search” to «view» “Issue Display” by invoking «controller» “filter by issue” from «view» “Issue Search”. The navigational behaviour is performed when the «controller» “filter by issue” is activated. As shown in Figure 8, the execute model of this «controller» is “invoke”. This means that the «controller»

will not be activated by the system automatically at times such as initiation; rather, it needs to be invoked by a user during interaction with the system.

6.3.4 NAVIGATION IN-THE-SMALL

A more detailed specification of UTSE-Guide navigation is shown in Figure 10. Three types of elements can be seen in this example:

- «view»: A navigational behaviour results in the change of «view»s. In the scenario studied here, “issue display” «view» is displayed to replace “issue search” «view».
- «controller»: As described earlier, the navigational activity is completed by one or many «controller»s, and the interconnections and execution sequence of these «controller»s are specified in the navigation diagram.
- Other resources: Other resources, such as legacy applications, databases, file servers and external Web sites, may be needed to perform the navigation. In this example, the «controller»s interact with “issues” database to obtain the information needed.

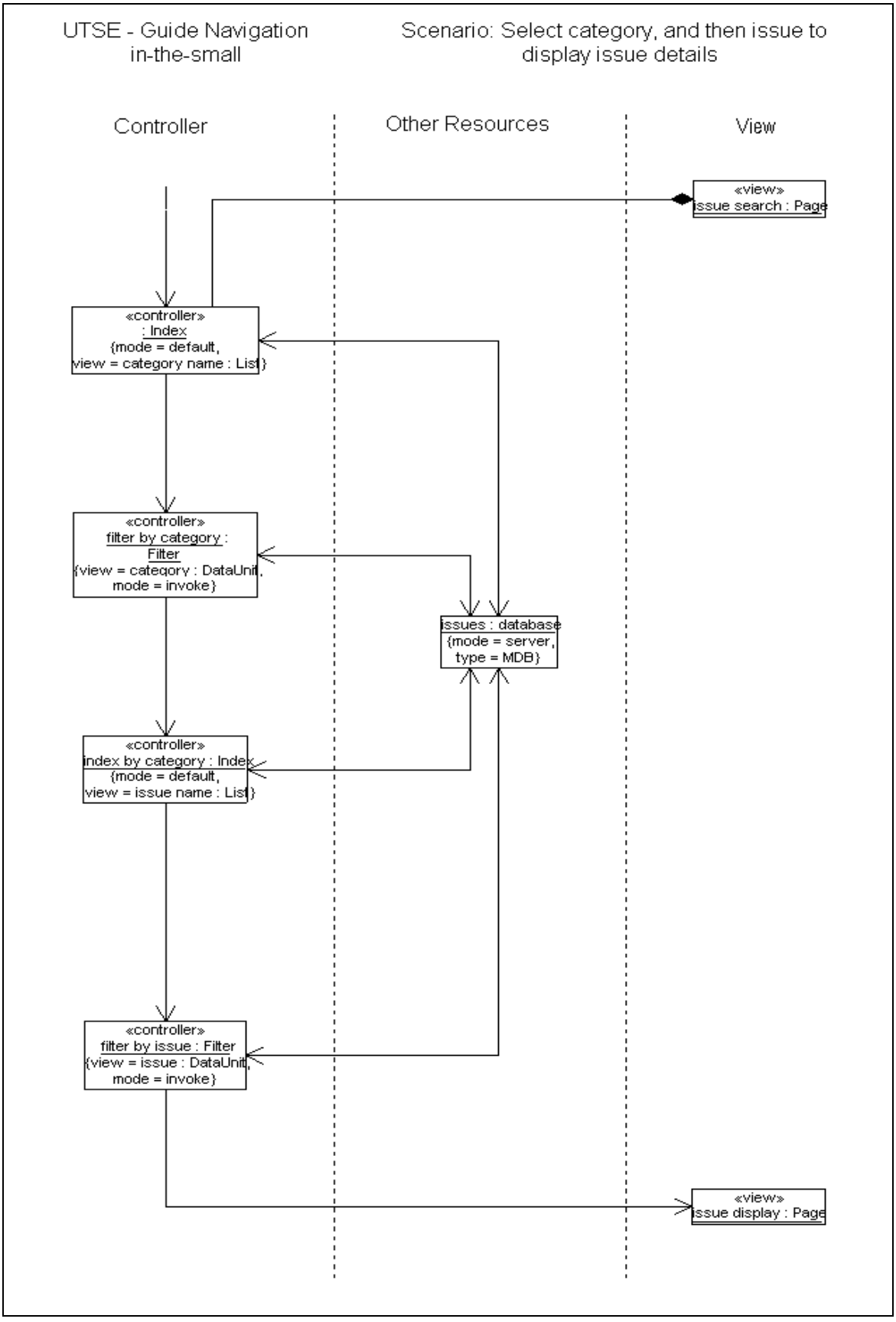
UTSE - Guide Navigation in-the-small
Scenario: Select category, and then issue to display issue details
Controller
Other Resources
View
«view»
issue search : Page
«controller»
: Index
{mode = default,
view = category name : List}
«controller»
filter by category :
Filter
{view = category : DataUnit,
mode = invoke}
issues : database
{mode = server,
type = MDB}
«controller»
index by category : Index
{mode = default,
view = issue name : List}
«controller»
filter by issue : Filter
{view = issue : DataUnit,
mode = invoke}
«view»
issue display : Page

Figure 10: Navigation Diagram – In-The-Small

6.4 PRESENTATION

A presentation diagram of the UTSE-Guide system is shown in Figure 11. It specifies the interface layer definition of the system. The presentational elements map to «view»s, and thus define the composition and format of «view»s when they are implemented in the user interface.

The flexibility of mapping each «view» to a number of «presentation»s provides the capability to implementation personalization, and also makes future change to the systems' presentational aspect relatively independent to the core architecture of the systems.

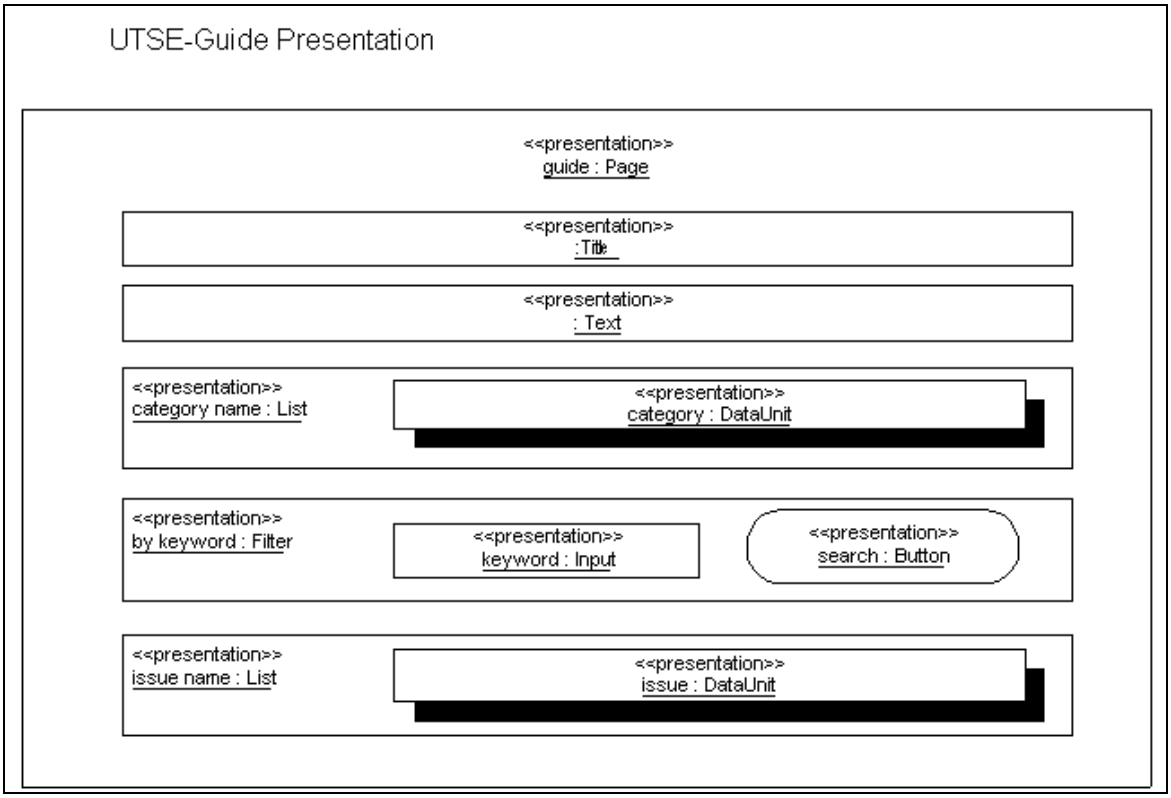

Figure 11: Presentation Diagram

6.5 FUNCTIONAL ARCHITECTURE

Due to the lack of sophistication in the UTSE-Guide system, no complex operation needs to be modelled separately using operation diagrams in this example. In fact, some simple operations are actually specified in the navigation diagrams.

6.6 POTENTIAL IMPROVEMENTS

As demonstrated by the case study, the proposed extensions to the UML notation can, if utilized properly, increase the modelling capability of the notation. These proposed extensions can be helpful in addressing the limitations in the existing modelling approaches that were identified in Gu *et al.* (2002). . An analysis of this potential is shown in Table 2.

Table 2: Extension Proposal – Addressing the Gaps

Gap in Existing Approaches	**Potential Improvements**
Inability to Model Sophisticated Functionality	The introduction of the Model-View-Controller concept provides the capability to model navigational and functional behaviours using the definitions of Controllers. Whilst some simple functions can be performed by individual Controllers, more sophisticated functions may need the collaboration of several Controllers. Some of these Controllers also support the integration to both internal and external applications and information resources. Whilst the functional modelling aspect cannot be fully demonstrated in this paper due to the lack of sophistication

	of the case study.
Disconnection between Functional Architecture and Information Architecture	The introduction of the Model-View-Controller concept and the extended model structure provides the potential capability to connect the functional architecture and the information architecture. Since the two aspects of Web system architecture are both connected to, and indeed developed from, the extended conceptual model, consistency and integrity are more likely to be achieved in the proposed approach.
Disconnection between Business Model and Technical Architecture	In the proposed extension model structure, the extended conceptual model is built upon the business requirements captured during requirements engineering. This extended conceptual model is then used as the foundation for the design of both the information architecture and the functional architecture of the Web system. This close connection introduced by the MVC structure can help to translate business requirements into the two aspects of Web system technical architecture.
Inability to Support Modelling at Various Abstraction Levels	With the modelling of composition, navigation and operation, diagrams can be constructed at two abstraction levels. For example, the composition of a Web system can be represented as logical elements at a high abstraction level, whilst it can also be modelled in term of pages and elements on the pages at a more detailed level. This is implemented by using the composition in-the-large and composition in-the-small diagrams. Although far from complete and thorough, this approach demonstrates the potential to address the issues related to modelling abstractions and interconnections between the abstraction levels.

Potential Misuse of UML Extension Mechanisms	Although this issue was not directly addressed in our proposal here, attention was given to the proposed UML extensions with the aim of avoiding the potential misuse of this mechanism.
Inability to Support System Life Cycle Management	This issue was not directly addressed. However, the proposed support for a more complete and balanced Web system technical architecture can potentially expand the usage of the model during the system life cycle.

7 SUMMARY

In this paper, we have proposed some extension directions to UML notation, with the aim of improving some aspects of the modelling language support for Web system development. The extensions to the conceptual model are based on the Model-View-Controller concept and the addition of several diagrams, such as composition diagrams, presentation diagrams, navigation diagrams and operation diagrams, can potentially increase the modelling capability offered by the existing UML notation.

Inspired by concepts taken from other modelling approaches, such as WebML, which support Web system information architecture reasonably well, and used in conjunction with the existing UML functional modelling capabilities, the proposed model structure aims to support both the functional and information architectures. The support for sophisticated functionality and the connection between the business model and the technical architecture is also addressed by the proposal. Furthermore, the proposed modelling approach can represent the system architecture at different abstraction levels and the linkage between these levels can be managed through the extended conceptual model.

A small case study was used to illustrate the proposed extensions and their applications although, due to the small size and complexity of the case study system, not all aspects and features of the extension proposal were fully demonstrated. Another, much larger case study has in fact been conducted using a real-world, commercially confidential, Web system, with excellent results.

References

Atkinson, C., 1998, "Supporting and Applying the UML Conceptual Framework", «UML»'98: Beyond the Notation, 1st International Workshop, Mulhouse, France, June 3-4, 1998, Selected papers, pp21-36

Atkinson, C. & Kühne, T., 2000, "Strict Profiles: Why and How", «UML»'2000: Advancing the Standard, 3rd International Conference, York, UK, October 2-6, 2000, Proceedings, pp309-322

Atkinson, C. & Kühne, T., 2001, "The Essence of Multilevel Metamodelling", in Proceddings of «UML»'2001 (the 4th International Conference on the Unified Modelling Language), 1-5 October 2001, Toronto, Canada

Atkinson, C., Kühne, T. and Henderson-Sellers, B., 2000, To meta or not to meta – that is the question, JOOP, 13(8), 32-35

Booch, G., Rumbaugh, J. and Jacobson, I., 1999, "The Unified Modelling Language User Guide", Addison-Wesley, pp482

Ceri, S., Fraternai, P. and Bongio, A., 2000, "Web Modelling Language (WebML): A Modelling Language for Designing Web Sites", Procs. WWW9/Computer Networks 33, 2000, 137-157

Gu, A., Henderson-Sellers, B. and Lowe, D., 2002, Web Modelling Languages: the gap between requirements and current exemplars, submitted for publication

Henderson-Sellers, B., Atkinson, C. and Firesmith, D.G., 1999, Viewing the OML as a variant of the UML, «UML»'99 –The Unified Modeling Language. Beyond the Standard (eds. R. France and B. Rumpe), LNCS 1723, Springer-Verlag, Berlin, Germany, 49-66

Hunt, J., 1997, "Smalltalk and Object Orientation: An Introduction", Springer, pp378

OMG, 2000, “OMG Unified Modeling Language Specification”, Version 1.3, June 1999, OMG document ad/99-06-09 [released to the general public as OMG document formal/00-03-01 in March 2000]. Available at http://www.omg.org

OMG, (2001). OMG Unified Modeling Language Specification, Version 1.4, September 2001, OMG document formal/01-09-68 through 80 (13 documents) [Online]. Available http://www.omg.org

Schwabe, D. and Rossi, G., 1998, “Developing Hypermedia Applications using OOHDM”, Workshop on Hypermedia Development Processes, Methods and Models (Hypertext’98), Pittsburgh, USA