WEB SERVICE INTERFACES DESIGN FOR E-BUSINESS APPLICATIONS

by

Sia Minh Hong

PhD Thesis

Presented to the Faculty of the Graduate School of

The University of Technology, Sydney

in Partial Fulfilment

of the Requirements

for the Degree of

Doctor of Philosophy in the

School of Software,
Faculty of Engineering and Information Technology

Supervisor: Dr. George Feuerlicht

October 2015

CERTIFICATE OF ORIGINAL AUTHORSHIP

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.

Signature of Student:

Date:

ACKNOWLEDGEMENTS

I would like to express my thanks to my principal supervisor, Dr. George Feuerlicht, for his guidance and support during the course of my PhD studies. He gave me much encouragement and ideas, which allowed me to complete this thesis. We worked together and I found his comments and feedback to be very useful.

I also would like to thank my co-supervisor Richard White, who is the CEO of Eagle Datamation International (EDI). He provided me with the opportunity to undertake this research work at the company on several occasions. I gained access to and obtained much valuable commercial experience when designing these Global Trade e-business Web Service interfaces.

Thanks are also extended to the project's funding or partner organisations: Australian Research Council (ARC Linkage Projects) and Eagle Datamation International (UTS contribution to ARC projects). This research thesis may not have happened without their support.

Thanks also go to my family and especially my wife for fully supporting and understanding what this long journey entailed. Thanks also go to Mr. Phillip Thomas who proof-read my thesis and provided me with advice regarding English corrections and expression.

Finally, my thanks go to the examiners for providing their valuable time for reading this thesis.

Table of Content

WEB SERVICE INTERFACES DESIGN FOR E-BUSINESS APPLICATIONS 1
LIST OF ACRONYMS & ABBREVIATIONS i

Abstract ..... **iii**

Chapter 1 ..... **1**

Introduction **1**
1.1 Background *1*
1.2 Web Services for e-business *3*
1.3 Web Service Interfaces *4*
1.4 Web Service Interface Design Issues *6*
1.5 Granularity Consideration *7*
1.6 Requirements for Web Service interface design *9*
1.7 Problem Identification and Definition *11*
1.8 Thesis Objective *12*
1.9 Summary of Contributions *14*
1.10 Research Methodology *15*
Evaluation 15
1.11 Structure of the Thesis *16*

Chapter 2 ..... **17**

E-Business Document Standards **17**
2.1 Web Service Invocation Methods *17*
2.2 Web Service with Business Document *19*
2.3 Normalisation of XML Messages / Business Document *21*
2.4 Business documents and their design pattern *22*
2.4.1 Business Object Document (BOD) BY OAGIS 22
2.4.2 Universal Business Language (UBL) BY OASIS 25
2.4.3 Open Travel Alliance (OTA) Business document 28
2.4.4 XML Common Business Language (xCBL) 31
2.4.5 GS1 Messages by global language of business 34
2.5 Command Pattern Interface *37*
2.6 Discussion *40*

Chapter 3 ..... **43**

Review of existing Web Service Interface Design, Business Document Design and Web Service Composition Design Methodologies **43**
3.1 Existing Web Service interface design methodologies *43*
3.1.1 Web Service design based on Elementary Business Function 43
3.1.2 Web Service design based on Requirement Analysis 45
3.1.3 Web Service Design based on Transactional Service 46
3.1.4 Web Service design based on Shareable Components 47
3.1.5 Web Service design based on Data Centric Approach with Factual Dependency 48
3.1.6 Generic Web Service Interface design methodology 48
3.1.7 REST style Web Service design methodology 49
3.1.8 Value-Based Service Modeling and Design 51
3.1.9 Model Driven Design of Web Service Operations using Web Engineering Practice 53
3.1.10 Pragmatic Web Service design approach 54

3.1.11 Evolving Web Service interface design ... 55
3.2 Business Document Design ... 56
3.2.1 Web Service design based on Document Engineering ... 56
3.3 Existing Web Service Composition design methodology ... 60
3.3.1 Web Service composition ... 60
3.3.2 Composition design based on UML ... 62
3.3.3 Composition Design based on Formal Language ... 63
3.3.4 Composition Design based on Case-Based Reasoning ... 64
3.3.5 Composition Design based on RosettaNet PIPs ... 64
3.4 Discussion ... 65

Chapter 4 ... 70

Proposed Methodology ... 70
4.1 Objective ... 70
4.1.1 Minimalist Interface Design ... 71
4.1.2 Methodology ... 71
4.1.3 Case study ... 72
4.2 Design with proposed solution ... 75
4.3 Advantages and Disadvantages ... 82
4.3.1 Advantages ... 82
4.3.2 Disadvantages ... 83
4.4 Discussion ... 84

Chapter 5 ... 87

Implementation ... 87
5.1 Implementing Case Study ... 87
5.1.1 Individual Business Process and Requirement ... 90
5.1.2 Proposed Ordering Process ... 94
5.2 Implementation Detail ... 95
5.2.1 System Architecture ... 95
5.2.2 Database System ... 96
5.2.3 Application server system ... 96
5.2.4 Application Prototype ... 97

Chapter 6 ... 100

Evaluation ... 100
6.1 Service-Oriented Design Principles ... 100
6.2 Quantitative Evaluation ... 103
6.3 Qualitative Evaluation ... 107
6.3.1 Reusability Principle ... 107
6.3.2 Business Suitability ... 110
6.3.3 Abstraction Principle ... 111
6.3.4 Extensibility ... 113
6.3.5 Flexibility ... 116
6.3.6 Maintainability ... 117
6.3.7 Comparison of the outcomes ... 119
6.4 Discussion ... 119

Chapter 7 ... 121

Conclusion ... 121
7.1 Research Summary ... 121
7.2 Research Contributions ... 124
7.3 Future Work ... 126

References **127**

Appendices **137**

Appendix 1: *Script to generate Database Table for case study* *137*
Appendix 2: *Available Web Method as Web Service* *139*
Appendix 3: *Persistence Unit for the Prototype Application* *140*
Appendix 4: *Business Entities of the Prototype application* *141*
Appendix 5: *Data Access Object of the Prototype application* *157*
Appendix 6: *Business Value Object of the Prototype application* *161*
Appendix 7: *Purchase Order Schema of the Prototype application* *183*
Appendix 8: *Object Serializer of the Prototype application* *185*
Appendix 9: *Prototype application Web Services* *192*
Appendix 10: *Prototype application Web Service Interface* *193*
Appendix 11: *Client's GUI of the Prototype application* *195*
Appendix 12: *Overview of the Project Prototype application* *201*
Appendix 13: *Web Service interface showing available ports and operations* *202*
Appendix 14: *Sample UBL 2.0 Purchase Order example (Source: OAGIS)* *203*
Appendix 15: *Sample UBL 2.0 Purchase Order Response example (Source: OAGIS)* *207*
Appendix 16: *Sample UBL 2.0 Change Purchase Order Response example (Source: OAGIS)* *210*
Appendix 17: *Sample UBL 2.0 Purchase Order Cancellation example (Source: OAGIS)* .. *229*

LIST OF ACRONYMS & ABBREVIATIONS

Abbreviation	
ABIE	Aggregate Business Information Entity
ACC	Aggregate Core Component
B2B	Business-to-business
BBC	Basic Core Component
BBIE	Basic Business Information Entity
BIE	Business Information Entity
BOD	Business Object Document
BPEL	Business Process Execution Language
CBL	Common Business Library
CCL	Core Component Library
CCT	Core Component Type
CCTS	Core Components Technical Specification
CEFACT	Centre for Trade Facilitation and Electronic Business
ebXML	Electronic Business eXtensible Markup Language
EDI	Electronic Data Interchange
EDIFACT	Electronic Data Interchange For Administration, Commerce and Transport
EPC	Electronic Product Code
GDD	Global Data Dictionary
GDT	Global data type
GS1	Global Standards One
HTTP	Hypertext Transfer Protocol
HTTPS	Secured HyperText Transfer Protocol
IATA	International Air Transport Association
ISO	International Organization for Standardization
MIME	Multipurpose Internet Mail Extension
NDR	Naming and Design Rules

OAGI	Open Applications Group, Inc.
OAGIS	Open Applications Group Integration Specification
OASIS	Organization for the Advancement of Structured Information Standards
OTA	Open Travel Alliance
OWL	Web Ontology Language
UBL	Universal Business Language
UDDI	Universal Description, Discovery and Integration
UMM	UN/CEFACT Modelling Methodology
WS-CAF	Web Services Composite Application Framework
WSDL	Web Services Description Language
xCBL	XML Common Business Library
XML	Extensible Markup Language
XSD	XML Schema
XSL	Extensible Stylesheet Language
XSLT	Extensible Stylesheet Language Transformations

ABSTRACT

As a result of the rapid developments in Web Service standards and technologies during the last decade, many organisations are implementing applications using Web Services. Some organisations are making significant commitments to Web Service standards and technology platforms. Successful projects using Web Services will to a large extent depend on the effective design and development methodologies used in the construction of an e-business application. While the importance of application design in general is recognised, so far only limited attention has been paid to design issues for service-oriented e-business applications. Currently there are no comprehensive methodologies for designing service interfaces.

The traditional e-business interoperability approach is for business partners to interchange the industry standard business documents or XML messages (i.e. UBL, OTA). This approach is complex and inefficient because the business document is large and results in many optional and repeated elements that are redundant. Developing Web Services for e-business is time-consuming due to the considerable effort required to define the interfaces and maintaining a large volume of standard business documents.

This work proposes to use the minimalist design approach to optimise a set of standard business documents and interfaces. The proposed interface is exposed as an abstract layer to the external parties and is able to process multiple actions corresponding to a business document. The method is based on analysing existing business documents, identifying the key elements responsible for an operation, and then exposing the operation interface that corresponds to the business document rather than the business event. This can be achieved by inserting business event action elements into the XML schema. Doing so will not only reduce the number of operation interfaces but also

increase the Web Service interface's flexibility and extendibility. Web Service implementation projects conducted in the absence of a design framework are likely to suffer from poor reuse and extensibility. In order to achieve reusability, this method enables the operation to be invoked based on individual action or multiple actions for the same interface. This is because a single action operation typically represents a fine-grained business task. Consequently, the interface is always extendable due to using multiple actions in the operation.

Finally, this thesis will detail the above mentioned methodology to optimise e-business Web Service interface to become more flexible, reusable and extendable. We will illustrate the design methodology using a purchase order business process example based on the Universal Business Language specification in order to demonstrate its effectiveness.

CHAPTER 1
INTRODUCTION

In this chapter, we first review the background of e-business particularly in the global trade context in Section 1.1. The Web Services and Web Service interfaces for e-business are described in Sections 1.2 and 1.3. In the next Section 1.4, we identify the problems concerning current Web Service interface design followed by Section 1.5, which discusses the granularity of the Web Services. Section 1.6 describes the requirements for Web Service interface design and following this, Section 1.7 will identify the problem and definition, while Sections 1.8 and 1.9 will specify the thesis objectives and contributions to the research. The thesis research methodology and its evaluation approach are described in Section 1.10 and is followed by the structure of the thesis in Section 1.11.

1.1 Background

Economic pressures are rapidly forcing businesses to automate e-business (electronic business) transactions. Successful e-business relies on accurate and secure interchange of information between different organisations, typically within specific vertical application domains such as travel, health and education. The principal challenge is to ensure e-business interoperability in an environment characterised by a large number of autonomous partner organisations with different technology platforms and business semantics. Traditional approaches such as using EDI (Electronic Data Interchange) (UNECE, 2004), or ebXML (ebXML, 2004) which use document interchange as the interoperability mechanism suffer from limited scalability (especially when the number of partner organisations increases) poor flexibility, and high implementation costs. The emergence of Web Service standards (WC3, 2004) and technologies and their wide

adoption by the IT (Information Technology) industry is an opportunity to solve the e-business interoperability challenge in the context of service-oriented computing. This research thesis is concerned with the application of service-oriented techniques to solving the problem of e-business interoperability, focusing on the design of service interfaces for domain-specific Web Service applications.

The importance of Web Service interface design is addressed in numerous publications. The challenge of this research is to develop an engineering method for Web Service interface design. This research focuses on seeking a design framework that will provide guiding principles to identify a consistent set of service interfaces. It can then be used to compose complex enterprise services in order to maximise reuse, avoid duplication and enable extendibility.

Traditional business approaches such as exchanging paper-based business documents between partners via fax or e-mail are slow, inefficient and costly. Electronic Business (e-business) on the other hand, is more suitable for the rapidly changing environment where any business requirement change has to be acknowledged and applied instantly. In this way, enterprises do not design, manufacture and sell products by themselves, but look for the best partners worldwide to establish alliances. They follow this approach in order to minimise costs and achieve their objectives in the shortest amount of time. These partners are distributed globally and their services can be changed and replaced by others. This transformation should not impact on the current information systems. An information system platform is a necessary part of the infrastructure for integrating an enterprise's partners so that they can exchange information and work together without time and costs pressures. Due to the characteristics of business processes in e-business, an information system platform of e-business has its own requirements. Firstly, the chosen information system platform must be suitable for the environment as enterprise systems for e-business are distributed geographically. Secondly, these systems must be an open platform. Since the enterprise partners are dynamic and distributed, the information system must be constructed in an open environment. Thirdly and lastly, the information system must be developed with loose coupling, where the enterprise partners are free to switch and change to others.

Web Services can be deployed on the Internet for distributed computing and transmission hence commercial business can be carried out together by several partners distributed geographically. Commercial services can be published by enterprises so that their business aims can take the form of Web Services. The integration of Web Services into e-business helps broaden the opportunity of business interaction between industry partners and the government sector. Applications can be developed based on standard Internet technology for distribution and integration. In particular with reference to e-business, Web Services have been designed and promoted to support both Business to Consumer (B2C) interaction and Business-to-Business (B2B) integration. More complex business applications can be expanded to each other's services.

Curbera (2001) defined a Web Service as a networked application that is able to interact with standard application-to-application web protocols over a well-defined interface, and is described as using standard functional description languages. A Web Service is essentially a semantically well-defined abstraction of a set of computational or activities involving a number of resources in order to fulfil some e-business requirements that are based on standardised XML messaging. Web Services allow applications to interact internally or externally; enterprise applications expose their business processes publicly as a service interface and are accessible on the Internet, for example: Google search service, Amazon online service and Flight Centre Booking Services. The success of these leading industry Web Services and the continued growth of others indicates that we will need a more standard and optimal abstraction Web Service interface to interoperate more effectively. Currently therefore, Web Service is the ideal candidate for integrating enterprise applications and setting up open and loose coupling information platforms for e-business.

1.2 Web Services for e-business

Web Services consist of a set of key technologies and standards that are suitable for e-business and can be developed so that it is capable of larger complex applications using existing technology and hardware. This makes it suitable to develop an e-business

application by reusing existing services. This can reduce costs and generate faster time to the market. It should be pointed out that there are still several issues including reusability, extendibility and flexibility that need to be addressed before the full potential of the e-business platform is realised concerning Web Service architectures.

Standards organisations such as OTA, OASIS, and UNEDIFACT design and maintain various domain-specific e-business specifications across vertical industries. These specifications such as the service interface and standard documents are essential for e-business systems to communicate in the same domain. As a result, each standards organisation will be required to define and maintain a large set of Web Service interfaces and business documents. A new version of the interfaces often takes months and years to be standardised and published (e.g. UBL version 1.0 released on 2004, UBL version 2.0 released in 2006 and UBL version 2.1 released in 2013). Both providers and clients need to revisit and design their interfaces to adopt the new version changes. This process is both complex and difficult; hence, upgradability and extensibility have become major hurdles in the e-business Web Service interface design issue.

1.3 Web Service Interfaces

Web Services provide a standard web application programming interface for e-business so that they can communicate on the Internet. A Web Service interface can be viewed as a universal Application Programming Interface (API) in which the interface is written in a platform independent manner. Service interfaces should cover all significant functions in the application domain. The number of operations provided should be adjusted carefully in order to decrease the complexity of application development. In order to create a good interface design, a set of design principles must be taken into account in the design methodology.

Web Services for e-business are typically based on a standard domain (i.e. UBL for standard Global Trade). Any providers and clients of the services must follow the specification guidelines in order to exchange the message. The business documents are

typically a collection of large coarse-grained complex schemas and they evolve over time. The business process operation is a business transaction that accepts a request message and produces a corresponding response message (i.e. purchaseOrderRequest and purchaseOrderResponse). Hence these request and response messages are typically coupled and will not be reused by others.

In traditional component oriented programming, method interfaces are separated from their implementation to provide an abstract definition; the developer can generate unique interface implementations using the same method. The method interfaces are the basic unit of reuse in an application program. In general, a program is an association of many small component interfaces: the contract that links the interfaces responsible for governing the parameter's exchange and the signature specifications for using that method. In contrast, Web Service interface is a logical grouping of method signatures that act as the contract between Web Service clients and Web Service providers. A Web Service composition is a combination of distinct Web Services to form a larger operation (e.g. a purchase order service can be combined with the payment service and the delivery service). Clients can locate and invoke any published Web Service interface and can even switch between different Web Service providers in a composition, as long as they are signature compatible. Very few changes are needed since a client only needs to modify the interface invocation.

In order to propose a methodology for the design of Web Service interfaces, in particularly for the Global Trade Domain, we will begin by exploring the fundamental properties of Web Service. Service-Oriented Architecture (SOA) is a platform for information exchange on the Internet and is becoming a leading paradigm for the development of information systems and application integration. With open and standard application interfaces and technologies, Web Services implement SOA's features to support loosely coupled applications and their integration. One of the main goals of SOA is to provide enterprise e-business solutions that can be extended or changed on demand while remaining compatible with an existing system (Papazoglou, 2003). The SOA is supposed to transform the web into a distributed business computation network (Lu, 2006). SOA bridges the gap between business and

information technologies by providing an approach for reengineering business processes with services oriented approaches. These solutions focus on the design of reusable, extendable Web Services and therefore, result in a well-defined interface since it provides a mechanism for integrating existing legacy applications with platform or language independently.

As Web Service interface design involves defining both the interfaces (i.e. service operation) and its payload (i.e. business document), essentially the key to a well-designed Web Service interface is to design an optimised standard document and its interface. Web Services for e-business tend to adopt coarse-grained messages (i.e. provide all required information in a single business document) due to a large collection of business objects contained in the document. This will reduce the amount of services invoked to fulfil a business process and hence minimise the number of requested and responses. Therefore, every business process will become a Web Service operation and consume one business document. As a result, the design of Web Service interfaces will involve analysis and modelling of the three abstraction layers in order to understand what will constitute an operation or a service. If we can move some of the business logics (i.e. processing instructions) to the business document, then we will be able to reduce the number of interfaces while maintaining the functionality. This thesis will provide a methodology in Chapter 4 to guide the design of transforming existing interfaces into a minimalist service interface in order to facilitate reusability and extensibility.

1.4 Web Service Interface Design Issues

There are several issues that need to be considered when designing a Web Service interface. Hui (2006) highlights three design concern areas. The first area of concern is the concept of development and choosing the granularity of the service. Every Web Service interface represents an abstraction model of a business process; the developer needs to understand and design the framework that matches the nature of the Web Service model. Furthermore, a developer choosing the right balance of granularity of a Web Service will provide a benefit to both the service provider and client. Decisions on

the level of service granularity should be based on the business requirement rather than a precise design approach. For instance, by adopting the multi-grained approach, a business document can be either a fine-grained, coarse-grained or mixed grain message depending on the business usage. The second concern area is that of maintenance and this refers to integration of services in order to maximise interoperability. The interface must be standardised across the domain and maintained by a standards organisation. For the Method Centric Approach, an initial research solution is the interface adapter framework, which uses an abstract interface on top to represent the service in a hierarchical relationship. For the Message Centric interface, these messages are developed and maintained by a few standard business alliances; this will help in promoting interoperability. However, they are designed for a specific domain in mind, and it is often not easy to integrate them with outside specific interactions. Another issue concerning maintenance is evolution of service, which represents the life span of the service interface. Some solutions include using a more flexible multiple version business document format instead of discrete input arguments. Alternatively, we can generalise and standardise the operational design so that the public can be used. The latter concern area is distributed environment and this refers to issues such as service error handling. Web Services for e-business are built on a distributed environment; we should consider this related issue during the design phase. Since network latency is an unavoidable problem, developers should consider re-requesting or resending the message during long network latency.

1.5 Granularity Consideration

Web Service granularity represents the abstraction level of a business process, and therefore it is important to find a balance between clients' needs and application requirements. Since Web Services have exposed the interface to the public, the decision on what constitutes an operation and the number of operations of a Web Service is a design issue. Web Services can have a coarse-grained, fine-grained or multi-grained interface.

Coarse-grained interface approach

The coarse-grained design approach focuses on higher-level abstraction and how the components interact within the larger business process context. In this approach, the client operations are generalisations of the corresponding server operations and hence the client interface can have fewer operations with less restrictive types. The advantage of coarse-grained design is that the client interfaces are not directly coupled to the service provider's interfaces, and each can evolve independently. The disadvantage is that the client may incur some performance cost due to dispatch large and complex messages. A coarse-grained interface is more suited for distributed processing and resource constrained thin-client operations.

Fine-grained interface approach

Fine-grained interface design results in operations that represent single tasks and lead to complex interaction dialogues. Consequently, the interface needs to be updated whenever the server interfaces changes. In addition, where a fine-grained service is changed, numerous related services also need to be updated because of high dependency. This is the major drawback of a fine-grained Web Service and can become a serious problem for distributed processing and large-scale deployment. The principal advantage of this approach includes reusability and the ability to reduce redundancy.

Multi-grained interface approach

In between coarse-grained and fine-grained interfaces, a developer can combine interface approaches to produce a multi-grained interface according to Stevens (2002). This is achieved by initially creating fine-grained services, and then wrapping them in coarse-grained facades, which create a multi-grained strategy. The author suggests that it is better to make fine-grained based services that access coarse-grained ones, as they are easier to deploy and manage. Smaller services will provide more options for their physical deployment.

1.6 Requirements for Web Service interface design

Web Service interface design is important because interfaces are the contract between the service provider and client. The quality and reliability of the interaction depend on the stability and effective design of the interface. Web Service interface design consists of tasks such as the selection of a service's operations, and their input and output parameters. This mostly depends on the fundamental design principle being applied. The Web Service protocol SOAP was designed as a simple way to undertake remote procedure calls (RPCs) over the Internet. A service designed with RPC-style interfaces will clearly separate operations and contain well-defined parameters. However, Document Style interface's design concentrates on the content of the messages in the system, and that the interfaces should minimise the set of operations. Eventually, this relies on the design of the interface's granularity. Service interface with optimal granularity will mostly be reusable and extendable. To date, there is no widely accepted design methodology for service interface design. We will discuss several existing Web Service designs and Composition design methodologies in the following paragraphs to illustrate the available support for them.

A Web Service can be viewed as a universal API in which the interface is expressed in a platform neutral manner. In order to have a good interface design, a set of design considerations/ principles must be borne in mind. These set of design considerations/ principles are supported by design methodology. According to Artus (2006), Bean (2009) and Erl, (2008) it is suggested that Web Services should be designed with flexibility in mind by increasing both loose coupling and ease of process implementation. Below is a summary of several design requirements:

i. **Loose Coupling**: The SOA design principle is to reduce dependencies between the client and the provider. A change made by the service provider should not necessitate corresponding changes in the client system; this includes platform, location, availability and versions. Loose Coupling helps to minimise the impact of changes to clients.

ii. **Consistency and Interoperability**: Across the application domain, Service interfaces should be uniformity defined (using the same rules for the names of Web Services, operations, input and output parameters) and consistently defined, i.e. list in the same order and apply the same pattern. This will greatly increase the interface's usability regardless of any technology platform; and therefore reduces development, integration and maintenance requirements.

iii. **Standard, formally defined interfaces**: Web Service interfaces should be defined with clarity and information should be defined to include the following: the task that the Web Service operation performs, its prerequisites and its outcomes, and the meaning of its parameters. Naming convention should be meaningful in the domain of expertise of the service client, where business concepts are chosen over technical concepts.

iv. **Maximise reuse**: Any Web Services should be intended to be reusable and can avoid focusing on the requirements of a service's initial clients. Instead, developers should determine a more complete requirement and consider the potential evolution of a service, which can evolve to respond to the new requirements in a controlled manner. This is because if the number of clients rises, requirements may well differ from the initial situation. Therefore, reusability can optimise the design and development process and thus reduce development time and costs.

v. **Well-chosen granularity**: This balances the number of operations a service should have. We can have few services with more operations or limited operations with more services. Depending on the Web Service, choosing the right granularity will maximise maintainability, operability and consumability.

vi. **Cohesive and Completeness**: Service interfaces should be developed in such a way that computing modules' internal elements are well linked to one another. This will increase stability because cohesion limits the scope of change in a specific service.

Service interfaces should cover all significant functions in the application domain. The number of operations provided should be adjusted carefully because interfaces with excessive or insufficient operations will increase the complexity of application development.

1.7 Problem Identification and Definition

A traditional e-business approach such as the Document Centric Model from Global Trade Domain focuses on interchanging documents between business partners. This is similar to the old-fashioned paper-based model; all requested data are encapsulated in a single coarse-grained XML business document. These standard documents are governed by explicit domain organisations (i.e. UBL, OTA and UN/EFIFACT). The principal advantage of this approach is the ability to seamlessly integrate across multiple platforms and is independent of the implementation languages. Business partners can easily communicate via these standard documents and interfaces. However, this approach requires complex and large sets of business processes and documents. Standardising these business documents and Web Service interfaces involve many organisations and requires considerable time and effort. This approach also suffers from poor reusability due to adopting coarse-grained operations and business documents (Feuerlicht, 2007). Each operation is coupled with a single business document and hence will impact on the extensibility since adding new operations and new elements require standardisation and release of a new version of the interface along with the corresponding business documents. Flexibility and reusability have also been neglected and not addressed in current methodologies (Feuerlicht, 2004). Furthermore, adopting these industry standard business documents is a complex task with many redundant elements. Numerous organisations are implementing B2B Web Services and some are making significant commitments to Web Service standard and technology platforms. The successes of projects using Web Services will to a large extent depend on the effective design and development methodologies used in the construction of e-business applications. As Web Services constitute basic building blocks of service-oriented applications, decisions about which operations should comprise a service, and what service interface should be exposed, are all of vital importance for e-business.

Ultimately they determine the quality and reliability of Web Service applications. Hence, a well-designed Web Service interface is a key requirement for ensuring a high level of interoperability in complex e-business applications. A good Web Service interface should be designed with a focus on reusability and any new functionality extension should have minimal impact (if any) of the existing interface. Web Service implementation projects conducted in the absence of a design framework are likely to suffer from poor reuse and extensibility. While the importance of application design in general is recognised, to date, only limited attention has been paid to design issues for service-oriented e-business applications. Currently, there are no comprehensive methodologies for designing of service interfaces.

1.8 Thesis Objective

The main objective of this research is to develop a methodology for the design of Web Service interfaces for e-business based on various literature reviews about existing e-business document-centric approaches. The overall aim is to develop reusable, extendable and flexible Web Service interfaces. An interface that will fulfil multiple business functions resulting in a minimal number of operations. This would lead to reduced impact of changes in the Web Service implementation or document structure that in turn improves stability of the interface and fosters loose coupling. A novel Web Service interface design methodology is proposed by combining the “command” software design pattern and generic interfaces which is also based on the principle of minimalism. According to this principle, we can simplify and optimise the Web Service interface to its minimal form (i.e. smallest set of methods) while keeping the required functionality. Following a top-down design approach, the design methodology will transform a set of e-business standard documents and interfaces into one well-optimised business document and service interface.

Specific Objectives:

The specific objectives of this research are as follows:

1 Propose a design method that will transform any existing Web Service interface into a more reusable service interface. Multi-grained operations will need to be supported to maximise reusability; individual fine-grained or coarse-grained operations should be reused by composition from and to other services. This is the basic requirement of any Web Services and conforms to the extensibility principle such that a new functionality can be added to the existing interface.

2 The transformed Web Service interfaces based on the design method should be extendable. The proposed minimalist design will make the interface more generic. Extension of new functionality is achieved by adding new actions with corresponding elements inside the business document. Updating any Web Service interfaces should not break any existing business integration. Service interface is a data contract agreed upon by many stakeholders while they are in development, and changing any elements should result in minimal impact (if any) to any service integration.

3 The resulting Web Service interfaces should also be flexible. Flexibility allows Web Services to incorporate multiple actions and thus minimise invocation calls. The proposed interface will allow multiple fine-grained operations to be processed in a single business document. This will permit sending only the required information and thus providing clear processing instructions to the provider.

4 Implementation and evaluation will be undertaken in order to assess the proposed interface design behaviour. The proposed Web Service interface based on the design method should conform to the service design principles.

1.9 Summary of Contributions

The main contribution of this thesis is the proposed design methodology for the Web Service interface design. Existing Web Service design methodologies mainly concentrate on the service design at the high or enterprise level. Our design methodology is concentrated on the design at a low level, i.e. Web Service interface. We will outline the design steps for developing the service interface and provide additional guidelines to map the interface to WSDL, to create a Web Service (Section 4.1.2). Furthermore, we will develop a Web Service application prototype to verify our proposed method (Section 4.2).

This research will contribute to Web Service design and development for e-business developers and companies. Developers can reuse Web Services rather than have to build a new application. For companies and users of Web Service applications, the proposed interface will be more reusable and extendable. Below are the contributions this thesis makes to the topic:

- This research will contribute a new approach to the design of Web Service interface and a new business document pattern (Section 4.1.2). The proposed service interfaces will be exposed based on business documents rather than business events. This can minimise the number of interfaces and therefore make maintenance easier. We can reveal a smaller number of interfaces while providing comparable or more aspects of the original functionality.

- The proposed interface will support multi-grained operation by adopting the command pattern design. The resulting interface and document will be optimised for reusability and extensibility.

- The proposed method will be developed and evaluated against the design principles using a simple Web Service application prototype based on the UBL specification.

- By defining an efficient and flexible service interface, Web Service applications will become more responsive and effective. This allows more application integration to be achieved across the e-business industry.

- As the proposed method is a generic interface design pattern, this will not only be applied to Web Service interface design but also to any other e-business interface design regardless of the implementation style, such as using the REST style.

1.10 Research Methodology

The design methodology is based on a top-down design approach by a transformation of the existing Web Service interface and related business documents. A survey of existing business document patterns and designs will be conducted in order to study the strength and limitation of the current e-business applications. Furthermore, an extensive literature review of existing Web Services and Web Service interface designs in both published articles and industry-based sources will be carried out in order to understand their limitations and issues. From the study of the existing design of service-oriented applications, we will propose a new engineering methodology for designing a Web Service interface by transforming the standard business document and its interface into a more reusable, extendable and flexible Web Service interface. This method will then be applied to design a consistent set of service interfaces for the Global Trade Domain, in particularly, based on UBL**.**

EVALUATION

An evaluation (Chapter 6) is also presented to assess the proposed Web Service interface design in comparison with the reference UBL business documents. In Section 6.2, we will carry out a quantitative approach evaluation by comparing the results from processing two different types of messages, as mentioned above. Multiple test case scenarios will be setup and done to examine the effectiveness of the proposed design document in contrast to the UBL document approach. In Section 6.3, a qualitative approach evaluation will be presented in order to verify that the proposed document

design conforms to the principles of the Service-Oriented Design developed by Legner (2007) and Erl (2008) in Section 6.1 including: Abstraction, reusability and business suitability. Each principle will be evaluated and compared using both the proposed document designs and the reference UBL e-Business documents. We will also extend and generalise the design framework to ensure applicability to other types of vertical domains.

1.11 Structure of the Thesis

The rest of this thesis is organised as follows. Chapter 2 will review several domain-specific e-business document standards and explain the concept of industry standard business documents and how they are designed. Several document design patterns also will be studied in this chapter and their advantages and disadvantages explained. Chapter 3 will review the existing Web Services and interface designs and identify their limitations. This chapter reviews the literature on existing Web Service interface design methods and several other composition design methods. Based on the findings of Chapters 2 and 3 we then present a comprehensive methodology for designing a Web Service interface for e-business in Chapter 4. That particular chapter explains the details concerning the proposed design method and how it will satisfy the objectives of this thesis. Chapter 5 describes the methodology's implementation with the domain-specific Web Service application (Global Trade Application). Chapter 5 also outlines the requirements for implementing the prototype application. Chapter 6 will verify the methodology for design issues: in-depth discussion and verification against the service design principles; and compare and benchmark the proposed methodology's results against the traditional UBL approach's results. Finally, Chapter 7 will conclude the thesis and provide suggestions for further research. This final chapter will also discuss the findings outlined in the objectives and how the proposed solution will satisfy the requirements.

CHAPTER 2
E-BUSINESS DOCUMENT STANDARDS

There are many standards organisations, for example OTA, OASIS, and UNEDIFACT that design and maintain different domain-specific e-business documents across vertical industries. These standard business documents are essential for e-business systems to communicate in the same domain. If every company develops and designs their own documents, interoperability will become very difficult due to complicated data mapping and errors. Web Service interface design involves defining both the interfaces (i.e. identifies operation) and its operation's payload (i.e. business document). Since e-business relies on exchanging a large number of documents, designing an optimised standard document and its interface are essentially the key to a well-designed Web Service interface. In this chapter, we will review several e-business document standards and particularly their relationship to their pattern design. This chapter will study several business document design patterns and focus on how the standard business documents are formed, their common elements library, the operation invocation styles and in particularly focussing on the extensibility and reusability of the elements and operations. Based on these findings, we will understand their limitations and constraints in the current business document design. We also study the differences in Web Service invocation methods, which then lead to a review of business document designs along with multiple industry standard practices.

2.1 Web Service Invocation Methods

We discussed the SOA and applications of Web Services for e-business in section 1.2. In this section, we will describe different Web Service invocation methods. There are two main types and these are: firstly, Remote Procedure Call (RPC); and secondly,

Document (DOC) Style, which refers to passing a document as the argument. We will compare the differences between these two styles and recommended one of them for e-business Web Services. Understanding the Web Service invoking techniques will help us design the message within Web Services so that it is more acceptable to the industry.

Web Service Description Language (WSDL) can be described using both styles, which are defined in the SOAP message. Remote Procedure Call (RPC) is similar to the traditional programming interface in that it passes one or more parameters as the argument for the Web Services. This style has been widely used on the Internet, for example flight booking and trip planning services. The RPC style consists of a fixed set of predefined elements that are embedded in the message. The advantage of RPC includes a fine-grained operation that is more reusable. The processing of the RPC message is faster because it is smaller and RPC use is easier since the elements are predefined in the operation. However, there are certain disadvantages in using RPC such as requiring additional information to maintain the state of the operation, and a complex business process that requires numerous invocations to fulfil the task. Another disadvantage of RPC is the tight coupling to the application method. When a method name or the parameters changes this will result in redefining the Web Service operation.

Since the industry is increasingly exchanging information between business partners, Document Style is the recommended way for communication in the Web Services (Vinoski, 2002). A business document is designed to be self-described, that is it should contain all the information needed to allow recipients to understand the context of the e-business process that the document supports. It is also relevant that it performs tasks without the need to refer to a previously sent document. Document Style relies on exchanging a business document to fulfil a business requirement. All the required business information will be put into the document in the form of a XML document. The XML document schema provides a template that includes all the necessary elements and their defined type.

The advantage of the Document Style is that it includes: firstly, definable complex data elements; secondly, less invocation calls by putting all the requested information in the

payload; thirdly, stated information can be maintained in the document; and fourthly, the XML document can be validated against the schema. Disadvantages of the Document-Centric approach include the complex structure of a document, which makes it difficult for it to be understood from a programming point of view. This leads to complexity in the development of translation software used to integrate e-business documents into the partner's internal systems. Document Style also tends to be less reusable due to its coarse-grained nature and specific business process development.

A Web Service might provide multiple versions of the interface by specifying the version in the WSDL. A Web Service might also process different versions of a business document by introducing the version number to the request and response schema when invoking the operation. This will maintain backward compatibility to any existing clients while the new interface is introduced. However, this may lead to multiple translators for each partner because of the above requirements. Several large organisations are using ESB (Enterprise Service Bus) for translating or adapting to interface versioning. ESB is a software middleware communication layer responsible for monitoring and control routing of the invocation messages.

Although there are industry standard groups such as UBL, UN/EdiFact and OTA (Open Travel Alliance) that promote their own standard business documents and public business processes, these standards are usually difficult to interpret and take a long time to implement. This is explained by the existence of large amounts of data and the requirement of participating companies to agree on the precise meaning of transmitted information, which is difficult to achieve (Feuerlicht, 2004). Consequently, if business conditions change, document standards need to be changed accordingly.

2.2 Web Service with Business Document

The industry tends to use coarse-grained messages (business document) instead of fine-grained ones to communicate between services. These business documents are normally domain-specific (e.g. Travel domain, Global Trade domain) and are maintained by a specific standards organisation (e.g. OTA, OASIS, UNEDIFACT). Their advantages

include: a business document contains all a query's information in a single XML file (e.g. a purchase order business document contains buyer, seller information, list of items and quantity), which are transferred only once. This will reduce the number of services invoked to fulfil a business process. The cost of transferring more data in a single document is marginal compared to the latency cost of going back and forth. This is particularly true for a large collection of business objects contained in the document. This simplifies the number of operations needed to interact with the application and minimises the number of requested responses. The developer only needs to design the document once and update it with a version number. XSD Schema makes this easy to construct by embedding types as elements within larger types. This will improve the interface's cohesiveness since a change in one element type will update all the inherited elements.

Web Services that consume business documents are normally coarse-grained operations. They focus on the client application's perspective and how the components interact within the larger business process context, which is where the two parties exchange their document (XML schema) to fulfil the business goal. These Web Service interfaces are exposed to the operations via the business process to fulfil a specific business task. Therefore, every business process will be equivalent to a Web Service's operation and consume one business document. It should be noted that one business process might contain many fine-grained operations calling simultaneously, i.e. a "purchase order" document may contain finer-grained operations such as: generating a new order ID, assigning date and time, customer database look up, currency converter and checking inventory where all these inter-related finer-grained operations are processed immediately at the end point. Consequently, the client interface requires fewer operations with less restrictive types since all the processes are done in one transaction. The advantage of a coarse-grained Web Service is loose coupling which is more suitable for distributed processing and a resource constrained thin client operation.

2.3 Normalisation of XML Messages / Business Document

Web Service interface uses XML message (Business Document) as their input/output parameter. Provost (2002) describes using the normalising method in relational database design and the application of the XML message. The author assumes that XML is the native language for data expression, and attempts to apply the concepts of normalisation to schema design. The goals of normalisation are to eliminate ambiguity in data expression, minimise redundancy, facilitate preservation of data consistency and enable rational maintenance of data. This concept can be applied from small to medium-sized messages with ease; however, if the message is very large and contains complex data types, the process will become very complicated and time-consuming to model and translate.

Arenas (2006) proposed the normalisation theory for XML. In this method, XFDs were defined in XML using the concept of "tree tuple" which in turn is based on the concept of the total un-nesting of a nested relation. Then a normal form for XML was defined (XNF) and an algorithm was presented, which converted an un-normalised XML document into one in XNF. Finally, the problematic implications for XFDs were investigated and complex results were provided for various classes of DTDs. Justification for XNF was also addressed. Using an approach based on information theory and entropy, it emerged that a slightly modified definition of XNF is a necessary and sufficient condition to maximise the amount of information stored in a XML document. The motivation for this normal form is similar to that of the Boyce-Codd Normal Form (BCNF) in a relational database, i.e. avoiding redundancy in database design. The standard decomposition technique produces a set of BCNF relations; this approach can convert a XML document into a set of documents in XNF.

Since a XML document has a hierarchical structure, representing XML in a tree tuple can enhance modelling elements and attributes instead of a flat structure as occurs in a relational database. Normalisation can further reduce the redundancy of the XML elements and attributes. However, this method can only apply to any single instance of XML. It is not concerned with the interaction between two exchange messages as in the nature of B2B interaction or the extensibility of the schema

2.4 Business documents and their design pattern

According to Hinkelman (2006) there are several basic business content design patterns. They have been adopted by various standards organisations to develop industry-specific business documents. According to Kabak and Dogac (2010), no single standard business document requirements satisfy all purposes of the all the e-business industries, as the requirements vary significantly different across different domains and geo-locations. The aim of the standard business document is to fulfil and adapt to a specific context, extensibility and customisation. This is despite the fact that most business document standards adopt the UN/CEFACT Core Component Technical Specification (CCTS) as the basis for describing the schema. According to Kabak and Dogac (2010), the UBL standard has been endorsed by governments throughout northern Europe such as the UK, Finland, Norway, Sweden, Iceland and Denmark as the official business documents for electronic government applications. Business Object Document (BOD) on the other hand receives the support of more than 40 countries and 38 industries. The majority of the BOD users are in the automotive vertical industry but also in the human resources, chemical and aerospace industries. GS1 documents are employed in more than 20 countries and across 20 industries. As a consequence, there is no particular reason for choosing one standard over another rather than by following what is considered to be conventional for a particular industry domain.

2.4.1 Business Object Document (BOD) by OAGIS

Business Object Document (BOD) or Business Content Envelope pattern proposed by OAGIS is a business document pattern designed to integrate business information with interaction and process indicators in the XML document. These are utilised in the design framework for Application-to-Application (A2A) or B2B. This pattern is considered more robust in that it takes on various processing abstractions. OAGIS uses this pattern to define the Business Object Document used for specifying business operations and transferring data between application systems. According to Stanley (2001) a BOD contains two parts: namely, Application area and Data area (Figure 2.1).

The Application area is similar to a message header, which contains application-specific information such as sender, time, date and code, which are common to all BODs. In contrast the Data area is similar to the payload, which contain the business data, referred as a “Noun”, and its associated actions are referred to as a “Verb” (Listing 2.1).

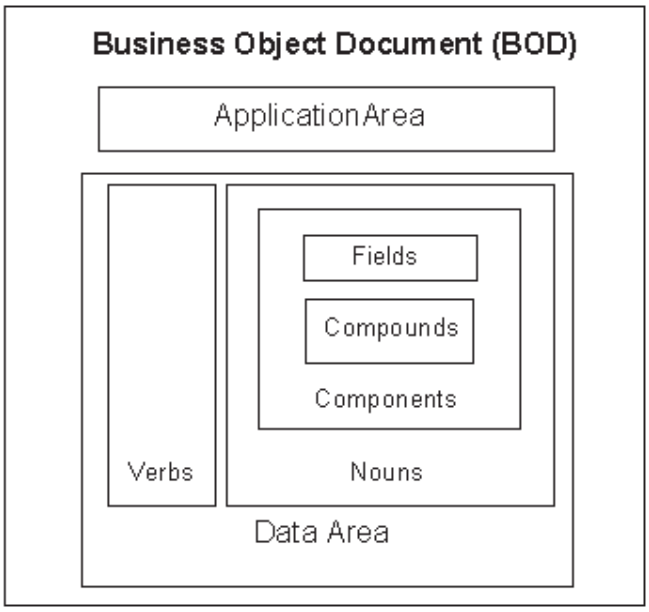


Figure 2.1 Layout of Business Object Document (Source: OAGIS)

Listing 2.1 Sample Business Object Document XML

```
<ApplicationArea>
   <CreationDateTime>2009-10-17T10:09:02.01Z
   </CreationDateTime>
   <BODID>PC2009-11-17-052</BODID>
</ApplicationArea>
<DataArea>
   <Process>
      <ActionCriteria>
         <ActionExpression actionCode="Add"> </ActionExpression>
         <ChangeStatus>
            <ReasonCode>New Purchase Order</ReasonCode>
         </ChangeStatus>
      </ActionCriteria>
   </Process>
</DataArea>
```

Web Service Interface	Business Function	Web Service Operation	Business Document
Purchase-Order.wsdl	CreateOrder()	AddPurchaseOrder()	AddPurchaseOrder.XML
	UpdateOrder()	UpdatePurchaseOrder()	UpdatePurchaseOrder.XML
	ChangeOrder()	ChangePurchaseOrder()	ChangePurchaseOrder.XML
	CancelOrder()	CancelPurchase()	CancelPurchase.XML
	NotifyOrder()	NotifyPurchaseOrder()	NotifyPurchaseOrder.XML
	ConfirmOrder()	ConfirmPurchaseOrder()	Confirm-PurchaseOrder.XML
	ResponseOrder()	Response-PurchaseOrder()	Response-PurchaseOrder.XML
	AcknowledgeOrder()	Acknowledge-PurchaseOrder()	Acknowledge-PurchaseOrder.XML
	GetListOrder()	GetListPurchaseOrder()	GetListPurchaseOrder.XML
	GetOrder()	GetPurchaseOrder()	GetPurchaseOrder.XML
	ListOrder()	ListPurchaseOrder()	ListPurchaseOrder.XML
	ProcessOrder()	ProcessPurchaseOrder()	ProcessPuchaseOrder.XML
	ShowOrder()	ShowPurchaseOrder()	ShowPurchaseOrder.XML
	SyncOrder()	SyncPurchaseOrder()	SyncPurchaseOrder.XML

Table 2.1 Summary of Business Object Document pattern Interface

Table 2.1 summarises all the relevant BODs for the PurchaseOrder business document. There are in total fourteen actions associated with this document that correspond to fourteen different operations. Because of the large number of operations supported by the business document, there will be many schemas that correspond to the operations, making it difficult to maintain and increase the complexity of the interaction.

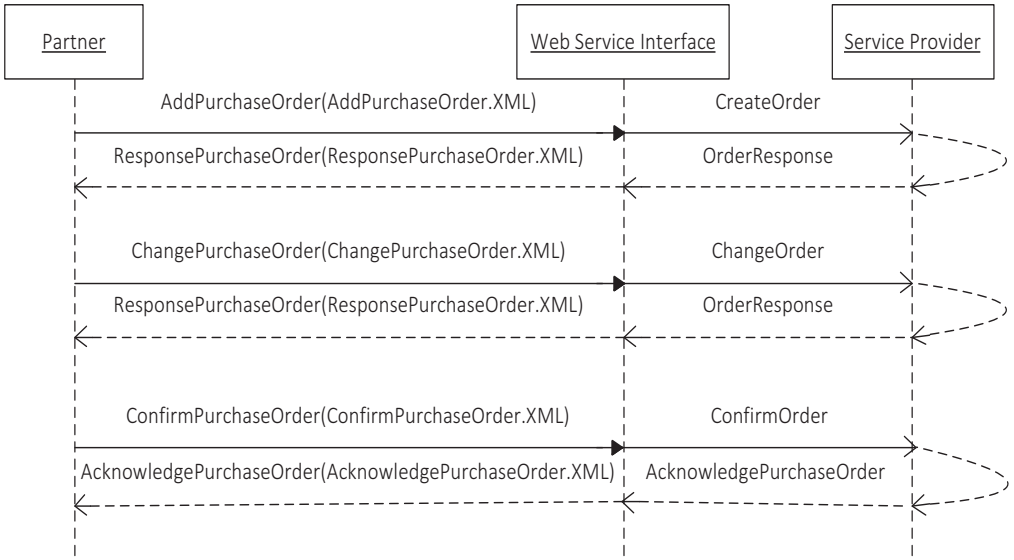


Figure 2.2 Sequence Diagram of Purchase Order Scenario designed using Business Object Document

For instance in Figure 2.2 (a "PurchaseOrder.wsdl" Web Service interface), "PurchaseOrder" is the "Noun" and followed by its associated actions/operations such as: "AddPurchaseOrder, DeletePurchaseOrder and CancelPurchaseOrder". For every operation, each will have a corresponding schema that processes the document depending on the Verb in the schema's data area. For example Operation "AddPurhaseOrder" will take an "AddPurchaseOrder.xsd" schema and the Verb will be "Add" in the data area even though the element "PurchaseOrder" is the same for every operation. Consequently, BOD reveals every possible operation with each action that has a corresponding schema. Extensibility is achieved by identifying new actions, adding extra operations and schemas.

2.4.2 UNIVERSAL BUSINESS LANGUAGE (UBL) BY OASIS

Universal Business Language (UBL) or Web Service-Based infrastructure pattern is designed based on a Web Service without concern for low-level infrastructure details. This is a "Usage-context free" pattern, which contains no action or processing instruction inside the document. According to Glushko (2005) the main focus is to define business processes based on an industry domain rather than where these processes define the business content. These business contents are intended to

accommodate both partners' requirements. One of the standards organisations adopting this pattern is OASIS, which developed the UBL. This framework is primarily focused on document exchange in B2B relying on the document as a common interface to retain a clean and stable relationship with its business partner despite some changes to its internal business processes. For example in a purchase order scenario the business process is identified first: PlaceOrder, ChangeOrder, CancelOrder and OrderResponse; then define a message for each business process. A "Place Order" Operation will process an "OrderRequest.XML" (Listing 2.2) and an "OrderChange" Operation will process an "OrderChange.XML". Since this framework concentrates on the business content, composing these messages is easier without concerns about how to handle the document. However, these documents do impose redundant data that are not necessary for some business interactions. For instance in Figure 2.3 to change an item in an Order, the developer must send an updated version of OrderChange.XML to accommodate the change.

Listing 2.2 Sample Universal Business Document OrderRequest XML

```
<ID>4500004875</ID>
 <IssueDate>2001-12-17</IssueDate>
 <BuyerParty>
  <ID>R300</ID>
  <PartyName>
   <Name>ABC</Name>
  </PartyName>
  <Address>
   <ID></ID>
   <Street>West Chester Pike</Street>
   <CityName>Sydney</CityName>
   <Country>
    <Code listID="3166-1" listAgencyID="ISO">AU</Code>
   </Country>
  </Address>
  <BuyerContact>
   <ID></ID>
   <Name>Joe Bloggs</Name>
  </BuyerContact>
 </BuyerParty>
 <OrderLine>
  <BuyersID></BuyersID>
  <Quantity unitCode="unit">10</Quantity>
  <Item>
   <ID>R100016</ID>
   <Description>Tuner X300</Description>
   <BasePrice>
    <PriceAmount currencyID="AUD">350</PriceAmount>
   </BasePrice>
  </Item>
 </OrderLine>
```

Web Service Interface	Business Function	Web Service Operation	Business document
Purchase-Order.wsdl	CreateOrder()	OrderRequest()	OrderRequest.XML
	ResponseSimpleOrder()	OrderResponseSimple()	OrderResponseSimple.XML
	ChangeOrder()	OrderChange()	OrderChange.XML
	ResponseOrder()	OrderResponse()	OrderResponse.XML
	CancelOrder()	OrderDelete()	OrderDelete.XML

Table 2.2 Summary of Universal Business Document Pattern Interface

Table 2.2 summarises of all the related "Order" business documents. There are total five different documents corresponding to five operations in the Web Service interface. Although this has fewer operations and documents than the BOD pattern, it includes all the essential business functions enough for B2B interactions since it is mainly designed for Web Service.

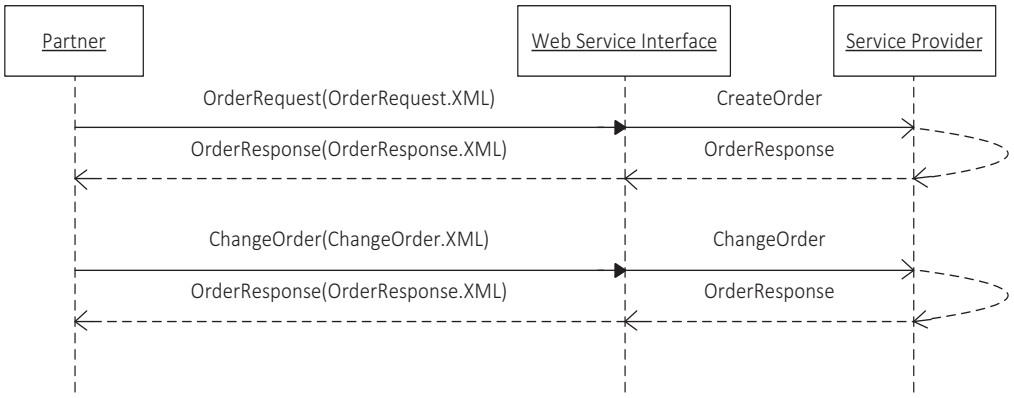


Figure 2.3 Sequence Diagram of Purchase Order Scenario designed using a Web Service-based infrastructure

As a result a UBL document does incur many redundant elements since updating a purchase order means sending an updated version of the order document.

2.4.3 OPEN TRAVEL ALLIANCE (OTA) BUSINESS DOCUMENT

Open Travel Alliance (OTA) business document or wrapped content pattern is designed based on wrapping the primary business content with a single style of interaction such as Request/Response. One standards organisation that adopted this pattern is OTA, which is a Global Travel domain-specific e-business document. The aim is to standardise all the activities concerning the travel industry. Every business process is designed based on Request/Response pattern for input and output (Figure 2.4). In addition to business content, organisations can define a common set of reusable process indicators that are built into request/response messages. These indicators are used to accommodate some usages that are common in a specific industry. For example, Echo Token, Status Code and transaction identifier serve to indicate the desired version of the payload response message. This Wrapped pattern is intended for Information query application where changing the business content is not a concern.

Business Process	Web Service Operation	Business document
FlightEnquiry	AirAvailRQ()	AirAvailRQ.XML
	AirAvaiRS ()	AirAvaiRS.XML
FlightBooking	AirBookRQ ()	AirBookRQ.XML
	AirBookRS ()	AirBookRS.XML

Table 2.3 Summary of Wrapped Content pattern Interface

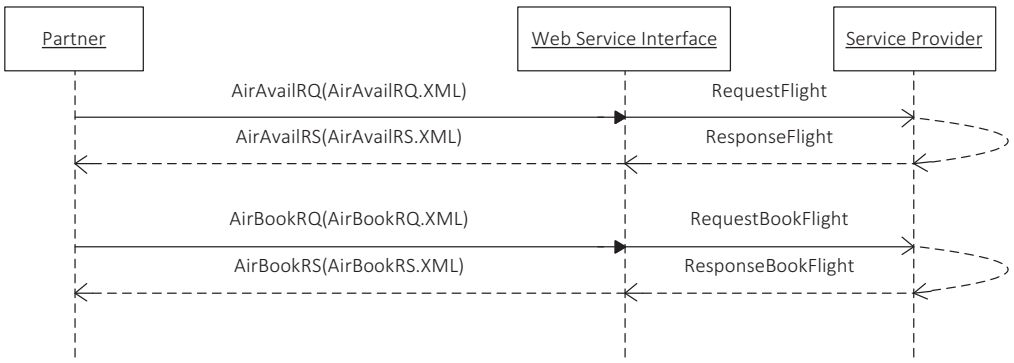


Figure 2.4: Sequence Diagram of Flight Enquiry Scenario based on Wrapped Content Pattern

Some modification business process documents from OTA such as AirBookChange, HotelBookChange and CarBookChange use the modificationType value (Listing 2.3). It indicates changes to the document and "actionType" to specify changes to each element.

Listing 2.3: ModificationType from OTA

Value	Description
1	Cancel entire booking file
2	Cancel partly
3	Name change - correction
4	Name change - new
5	Other
6	Split only
7	Split and update
8	Delete and add name

Listing 2.3 represents the ModificationType Code list derived from the OTA specification. These Code lists can be used within any modification document such as airbook, hotelbook, and carbook.

Listing 2.4 below represents the generic operation applied to any modification document in OTA. These actions give the instructions to the service provider on how to process the corresponding elements.

Listing 2.4: ActionType from OTA

Value	Description
Add-Update	Typically used to add an item where it does not exist or to update an item where it does exist.
Add	Typically used to add data whether data already exists or otherwise.
Cancel	Typically used to cancel an existing item.
Replace	Typically used to overlay existing data.
Delete	Typically used to remove specified data.

Listing 2.5: Fragment of a Modification of a traveller within an airBookModifyRQ Document [Source: OTA]

```
<AirBookModifyRQ ModificationType="5">
  <TravelerInfo>
    <AirTraveler PassengerTypeCode="ADT">
      <PersonName>
        <GivenName>Bertram</GivenName>
        <Surname>King</Surname>
        <NameTitle>Mr</NameTitle>
      </PersonName>
      <Telephone PhoneLocationType="5" PhoneTechType="1" PhoneNumber="043 317 36 75" Operation="Delete" />
      <Telephone PhoneLocationType="5" PhoneTechType="1" PhoneNumber="01 812 00 73" Operation="Add" />
      <Email EmailType="2" Operation="Delete">b.king@eemail.com</Email>
      <Email EmailType="2" Operation="Add">b.king@email.com</Email>
      <Address FormattedInd="false" Type="1" Operation="Delete">
        <AddressLine>Hauptstrasse 37b</AddressLine>
        <AddressLine>8302 Kloten</AddressLine>
        <AddressLine>Switzerland</AddressLine>
      </Address>
      <Address FormattedInd="false" Type="1" Operation="Add">
        <AddressLine>Zeltweg 3a</AddressLine>
        <AddressLine>8302 Kloten</AddressLine>
        <AddressLine>Switzerland</AddressLine>
      </Address>
      <CustLoyalty ProgramID="LX" MembershipID="234234343" LoyalLevel="Platinum" Operation="Add"
VendorCode="LX" />
      <TravelerRefNumber RPH="1" />
    </AirTraveler>
  </TravelerInfo>
</AirBookModifyRQ>
```

Listing 2.5 shows an example of modification details concerning a passenger in an airbook document. The “modificationType” is “5” which refers to “other” changes. To change the “phoneNumber”, “email”, and “address”, the sender must send an action type “delete” to the old information and send action type “add” with the new information. More information can also be added by using an action type “add”, for instance “cusLoyalty” program.

From the above example (Listing 2.5), OTA applies different Code Values (Listing 2.3) as well as Action Types (Listing 2.4) to an AirBookModify Request Message. OTA uses these values to indicate to the receiving end on how to deal with these intermittent elements. This is an illustration of using the command pattern (see Section 2.5) where commands are passed as parameters for executing instructions. However, OTA’s use of the Command pattern approach is limited to “modify request” messages and these do not apply to all messages.

2.4.4 XML COMMON BUSINESS LANGUAGE (XCBL)

XML Common Business Language (xCBL) is a set of standard XML business documents that facilitates global trading in e-commerce and version 4.0 is currently the latest revision. These documents are designed and built based on the standard UBL specification, including a common business library such as description of businesses and individuals, measurements, date, time and codes. The design principle's aims are to model the information requirements for business to business, enabling interoperability and modular composition of XML documents from the standards and reusable building blocks, and supports a variety of programming models. Apart from some standard business documents such as orderRequest, AvailabilityCheck, and PriceCheck, xCBL also supports multiple changes or cancel line items within an orderChange XML document.

These are a few example XML documents from xCBL. Listing 2.6 shows an item detail from an original xCBL order document. To send a new order, the sender needs to fill in every necessary element.

Listing 2.6. Fragment of an item detail within an Order Detail of an original Order Document [Source: xCBL]

```
<ItemDetail>
  <BaseItemDetail>
    <LineItemNum>
      <core:BuyerLineItemNum>00011</core:BuyerLineItemNum>
    </LineItemNum>
    <ItemIdentifiers>
      <core:PartNumbers>
        <core:SellerPartNumber>
          <core:PartID>R-3456</core:PartID>
        </core:SellerPartNumber>
        <core:BuyerPartNumber>
          <core:PartID>R-3456</core:PartID>
        </core:BuyerPartNumber>
      </core:PartNumbers>
    </ItemIdentifiers>
    <TotalQuantity>
      <core:QuantityValue>1</core:QuantityValue>
      <core:UnitOfMeasurement>
        <core:UOMCoded>EA</core:UOMCoded>
      </core:UnitOfMeasurement>
    </TotalQuantity>
  </BaseItemDetail>
  <PricingDetail>
    <core:ListOfPrice>
      <core:Price>
        <core:UnitPrice>
          <core:UnitPriceValue>1000.00</core:UnitPriceValue>
          <core:Currency>
```

```
                <core:CurrencyCoded>USD</core:CurrencyCoded>
              </core:Currency>
            </core:UnitPrice>
            <core:CalculatedPriceBasisQuantity>
              <core:QuantityValue>1</core:QuantityValue>
              <core:UnitOfMeasurement>
                <core:UOMCoded>EA</core:UOMCoded>
              </core:UnitOfMeasurement>
            </core:CalculatedPriceBasisQuantity>
          </core:Price>
        </core:ListOfPrice>
        <core:LineItemTotal>
          <core:MonetaryAmount>1000.00</core:MonetaryAmount>
        </core:LineItemTotal>
      </PricingDetail>
      <DeliveryDetail>
        <core:ListOfScheduleLine>
          <core:ScheduleLine>
            <core:Quantity>
              <core:QuantityValue>1</core:QuantityValue>
              <core:UnitOfMeasurement>
                <core:UOMCoded>EA</core:UOMCoded>
              </core:UnitOfMeasurement>
            </core:Quantity>
            <core:RequestedDeliveryDate>2001-02-07T00:12:00</core:RequestedDeliveryDate>
          </core:ScheduleLine>
        </core:ListOfScheduleLine>
      </DeliveryDetail>
      <LineItemNote>Item text manually entered</LineItemNote>
      <ListOfStructuredNote>
      </ItemDetail>
```

Listing 2.7: Fragment of a line item within a ChangeOrder Detail of a ChangeOrder Document [Source: xCBL]

```
<ChangeOrderDetail>
  <ListOfChangeOrderItemDetail>
    <ChangeOrderItemDetail>
      <ItemDetailChangeCoded>QuantityIncrease</ItemDetailChangeCoded>
      <OriginalItemDetailWithChanges>
        <BaseItemDetail>
          <LineItemNum>
            <core:BuyerLineItemNum>00011</core:BuyerLineItemNum>
          </LineItemNum>
          <TotalQuantity>
            <core:QuantityValue>5</core:QuantityValue>
            <core:UnitOfMeasurement>
              <core:UOMCoded>EA</core:UOMCoded>
            </core:UnitOfMeasurement>
          </TotalQuantity>
        </BaseItemDetail>
      </OriginalItemDetailWithChanges>
      <LineItemNote>Change quantity from 1 to 5.</LineItemNote>
    </ChangeOrderItemDetail>
  </ListOfChangeOrderItemDetail>
</ChangeOrderDetail>
```

Listing 2.7 is an example demonstrating how to change a line item, where we only need to specify the BuyerLineItemNum (i.e. 00011), itemDetailChangeCoded (i.e. QuanityIncrease) and to specify well the reason in LineItemNote (i.e. Change quantity from 1 to 5). xCBL supported other changes as well such as Price, Quantity, Delivery Date, Address, Shipping and Replaced. This approach provides flexibility and

extensibility to the document since the ChangeCoded can be added or deleted without affecting the business document. Senders can use a generic action “other” in Listing 2.8 to specify any instruction in the “LineItemNote” that does not belong to any of the Code lists.

xCBL design is another example of using the command pattern (see Section 2.4) to use parameters as execution instructions. However, this approach still requires multiple documents such as order, orderChange and orderResponse, and the ChangeCoded lists (Listing 2.8) are long and complicated. For instance, if a user makes changes to the date for delivery, the value change code of “ChangeOfDates” will be used in the change order. The ChangeOrder Document also includes repeated and redundant elements such as the original order details.

Listing 2.8: Detailed Change code list from xCBL [Source: xCBL]

```
<xsd:enumeration value="Other">
<xsd:enumeration value="AbideOutcomeOfNegotiations">
<xsd:enumeration value="AddAdditionalItems">
<xsd:enumeration value="Audited">
<xsd:enumeration value="BuyerClaimsAgainstInvoice">
<xsd:enumeration value="BuyerHasDeductedAmount">
<xsd:enumeration value="CallOffDelivery">
<xsd:enumeration value="Cancelled">
<xsd:enumeration value="ChangeOfDates">
<xsd:enumeration value="ChangeOfDateTerms">
<xsd:enumeration value="ChangesToItemLevelAllowanceOrCharges">
<xsd:enumeration value="ChangesToLineItems">
<xsd:enumeration value="ChangesToTerms">
<xsd:enumeration value="ChargeBackToSeller">
<xsd:enumeration value="Closed">
<xsd:enumeration value="ClosedAfterReopening">
<xsd:enumeration value="ConditionallyPaid">
<xsd:enumeration value="CorrectionOfError">
<xsd:enumeration value="EquipmentProvisionallyRepaired">
<xsd:enumeration value="ItemDeleted">
<xsd:enumeration value="ItemNumberChanged">
<xsd:enumeration value="New">
<xsd:enumeration value="NoAction">
<xsd:enumeration value="NoDelivery">
<xsd:enumeration value="NotPaidPredeterminationPricingOnly">
<xsd:enumeration value="OnAppeal">
<xsd:enumeration value="PreviousPaymentDecisionReversed">
<xsd:enumeration value="PriceChanged">
<xsd:enumeration value="Proposed">
<xsd:enumeration value="ProposedAmendment">
<xsd:enumeration value="QuantityDecrease">
<xsd:enumeration value="QuantityIncrease">
<xsd:enumeration value="Reaudited">
<xsd:enumeration value="Redetermined">
<xsd:enumeration value="ReferredIemRejected">
<xsd:enumeration value="ReferredItemAccepted">
<xsd:enumeration value="Reopened">
<xsd:enumeration value="ReplaceAllDates">
<xsd:enumeration value="ReplaceAllValues">
<xsd:enumeration value="Replaced">
<xsd:enumeration value="ReplacementItem">
```

```
<xsd:enumeration value="ReplacementItemWithModifications">
<xsd:enumeration value="ReplaceModeOfShipment">
<xsd:enumeration value="Reschedule">
<xsd:enumeration value="Reschedule-QuantityChange">
<xsd:enumeration value="Reviewed">
<xsd:enumeration value="Schedule">
<xsd:enumeration value="SellerWillIssueACreditNote">
<xsd:enumeration value="TermsChangedForNewTerms">
<xsd:enumeration value="TransferItem">
<xsd:enumeration value="UnitPrice-QuantityChange">
<xsd:enumeration value="UnitPrice-RescheduleChange">
<xsd:enumeration value="ChangeToTotalLevelAllowanceOrCharge">
<xsd:enumeration value="ConcurrentItemNoChange">
```

2.4.5 GS1 MESSAGES BY GLOBAL LANGUAGE OF BUSINESS

GS1 XML is a set of standard XML business documents designed to allow automatic electronic transmission of data. GS1 is part of eCom (Electronic business messaging standards) and is a component of the GS1 System. GS1 XML document consists of three layers, these being the transport layer, service layer and business layer. The transport layer contains information for routing and processing of the XML document. The service layer provides instructions or commands for the receiver to perform the action on the documents. The business layer contains the business document with all the required values to fulfil business requirements. The advantages of GS1 XML include a smaller number of business documents. Different commands can be supported in the same document. Therefore, instead of published multiple business documents such as “add order”, “change order” or “delete order”, different commands can be applied to the order document such as “add”, “change” and “delete”. Furthermore, the same commands can be reused on different documents and consequently, this means adding new functionality without affecting existing document definitions. Additional values can be added to the command lists and used by interested parties only, while others remain unchanged. Typical commands used in GS1 are add, change by refresh, correct and delete. For instance, to send a new order (Listing 2.9 - line 1) we need to use the command “add” along with the order document. To update an order (Listing 2.10), we need to employ the command “change by refresh” or “correct” with the modified version of the order document. To cancel an order (Listing 2.11), we are required to send the command “delete” along with the order document. GS1 also supports batch processing, which contains multiple commands and multiple transactions.

Listing 2.9: Sample GS1 new order document [Source: GS1]

```
<documentCommandHeader type="ADD">
  <entityIdentification>
    <uniqueCreatorIdentification>246810N</uniqueCreatorIdentification>
    <contentOwner>
      <gln>8712345678920</gln>
    </contentOwner>
  </entityIdentification>
</documentCommandHeader>
<documentCommandOperand>
  <order:multiShipmentOrder documentStatus="ORIGINAL" creationDateTime="2009-08-07T09:00:00Z">
    <multiShipmentOrderLineItem number="1">
      <requestedQuantity>
        <value>15</value>
      </requestedQuantity>
      <tradeItemIdentification>
        <gtin>88123456798906</gtin>
      </tradeItemIdentification>
    </multiShipmentOrderLineItem>
    <multiShipmentOrderLineItem number="2">
      <requestedQuantity>
        <value>20</value>
      </requestedQuantity>
      <tradeItemIdentification>
        <gtin>88123456798760</gtin>
      </tradeItemIdentification>
    </multiShipmentOrderLineItem>
  </order:multiShipmentOrder>
</documentCommandOperand>
```

Listing 2.9 represents a new order using GS1 XML with 2 line items. Line item 1 contains 15 and line item 2 contains 20. A new order is represented in the command header with type = “add” along with all the required information. Listing 2.10 represents an updated order using GS1 XML with 2 line items. This revised update changes line item 1 to a quantity 10, and keeps the line item 2 as unchanged. A change order is represented in the command header with type equal to “change by refresh” along with all the updated information. Basically it means overwriting with the latest information.

Listing 2.10: Sample GS1 change order document [Source: GS1]

```
<documentCommandHeader type="CHANGE_BY_REFRESH">
  <entityIdentification>
    <uniqueCreatorIdentification>246820M</uniqueCreatorIdentification>
    <contentOwner>
      <gln>8712345678920</gln>
    </contentOwner>
  </entityIdentification>
</documentCommandHeader>
<documentCommandOperand>
  <order:multiShipmentOrder documentStatus="ORIGINAL" creationDateTime="2009-08-07T09:45:00Z">
    <multiShipmentOrderLineItem number="1">
      <requestedQuantity>
        <value>10</value>
      </requestedQuantity>
      <tradeItemIdentification>
        <gtin>88123456798753</gtin>
      </tradeItemIdentification>
    </multiShipmentOrderLineItem>
```

```
      <multiShipmentOrderLineItem number="2">
        <requestedQuantity>
          <value>20</value>
        </requestedQuantity>
        <tradeItemIdentification>
          <gtin>88123456798760</gtin>
        </tradeItemIdentification>
      </multiShipmentOrderLineItem>
    </order:multiShipmentOrder>
</documentCommandOperand>
```

Listing 2.11 represents a cancel order using GS1 XML with 2 line items. Line item 1 contains 15 and line item 2 contains 20. A cancel order is represented in the command header with type = “delete” along with all the required information.

Listing 2.11: Sample GS1 cancel order document [Source: GS1]

```
<documentCommandHeader type="DELETE">
  <entityIdentification>
    <uniqueCreatorIdentification>246830C</uniqueCreatorIdentification>
    <contentOwner>
      <gln>8712345678920</gln>
    </contentOwner>
  </entityIdentification>
</documentCommandHeader>
<documentCommandOperand>
  <order:multiShipmentOrder documentStatus="ORIGINAL" creationDateTime="2009-08-07T09:45:00Z">
    <multiShipmentOrderLineItem number="1">
      <requestedQuantity>
        <value>15</value>
      </requestedQuantity>
      <tradeItemIdentification>
        <gtin>88123456798906</gtin>
      </tradeItemIdentification>
    </multiShipmentOrderLineItem>
    <multiShipmentOrderLineItem number="2">
      <requestedQuantity>
        <value>20</value>
      </requestedQuantity>
      <tradeItemIdentification>
        <gtin>88123456798760</gtin>
      </tradeItemIdentification>
    </multiShipmentOrderLineItem>
  </order:multiShipmentOrder>
</documentCommandOperand>
</eanucc:documentCommand>
```

From these above examples, we can observe the flexibility when using GS1 documents. However, the disadvantage of GS1 XML document is that of limited commands and the fact that it contains redundant elements. For instance to update an order, we are required to send the revised version of the order document, and to cancel the order we need to send the entire order document.

2.5 Command Pattern Interface

Other design patterns that were mentioned previously all follow the pattern where one operation processes a single message (i.e. a one to one relationship between the business document and its operation, an operation will not process other documents). In contrast to Command design patterns, this pattern supports processing instructions that are embedded in the request message. For instance, many operations can be combined into one operation while producing the same result (i.e. many to one relationship between the operation and its business document). This will not only provide a simpler unique interface but also provide extensibility by the action terms. Consequently, we chose to study the command pattern due to the support of transactional behaviour, its simplicity and practicality.

Command pattern (Object behaviour pattern) applies to different kinds of controllers that share a simple common interface, and commands are often referred to as an "action". Command pattern can be defined as encapsulating a request as an object, thereby letting the user parameterise clients with different requests, queue or log requests and support undoable operations (Erich, 1995). A command in a Command pattern is a single object that is passed by the client along with the message to the receiver in order to evaluate the message accordingly.

A command can contain one or more actions in the message. It can be placed and stored in the queues for invocation, and subsequently be dynamically modified to vary the receiver or message parameters. Each command has a common interface that contains the performAction() method. The command object stores the action information and applies the actual change when the performAction() method is called. This permits a decoupling of the invoker of the command and the handlers of the command (Crawford and Kaplan, 2003).

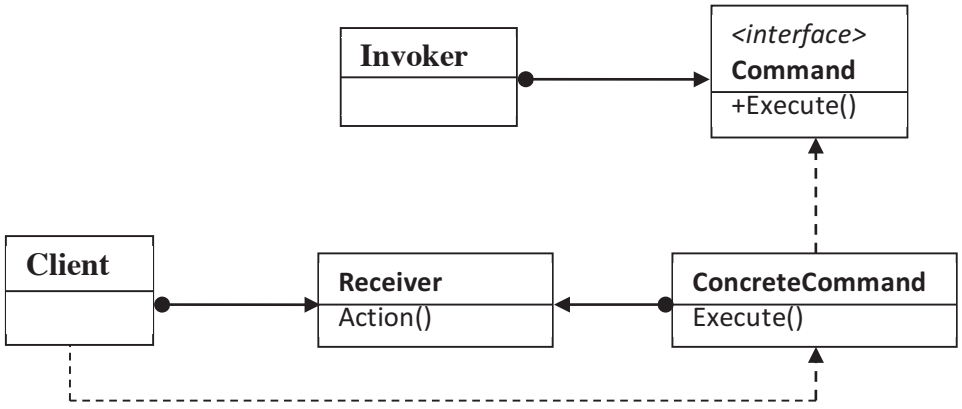


Figure 2.5: Command Pattern Interface

In Figure 2.5 the Invoker (the user of the interface) can invoke the object (command) execute() method to use the object's service. The ConcreteCommand (abstract base class) is responsible for mapping the action in the object to the receiver (a pointer to an operation). The Receiver will receive the action and process the operation(). The command classes were executed remotely in a command executor, located in the application's business layer. Therefore the command decouples the objects that invoke the operation from the one who knows how to execute it. The sequence of command (combined command) objects can be assembled into composite commands for the remote call. This utilises the flexibility of the command design pattern and saves as many remote calls to the business layer as possible.

According to Crawford and Kaplan (2003) the Command pattern has several advantages. It has inherent reusability, the interface and actions can be reused to execute similar operations, and they refer to the same object. This also increases reuse of classes by decoupling the interface from the implementation that provides weak coupling between clients and systems. This enables a developer to vary the behaviour of a class independent of its context. The Command pattern interface is highly extensible because it can employ any action object in the interface. Furthermore, since commands are passed as objects in the interface, it can be stored in the queues and applied later. It stores all the data necessary for a particular request. This is a typical example of a transactional operation, which often consists of various steps. If any step fails or due to the user cancelling it at any point, it is necessary to roll back any changes that occurred. Therefore, it supports undoable operations by reversing the command actions and

undoing whatever changes it made. Commands interface also supports run time association of a request to an object and is known as dynamic binding. Dynamic binding means that the execution of operations is determined at runtime. Therefore any object that has the correct interface will accept the request.

Command pattern in Web Services has been addressed and outlined in Java design Web Service guidelines (Java Blueprint, 2004). In order to have multiple methods for each document type such as "submitSupplierInvoice", "submitBillingInfo", the Web Service interface has a common method known as "submitDocument" that receives all document types (Figure 2.6). The receiver will map these documents to the processing logic that handles them. This means using commands to manage and identify all schemas of the various document types. Adding a new document is simply adding a new command to handle the new schema. The interface will not change to accommodate new document types, only the service implementation changes. This is because the command mappings are located in the interaction layer instead of the business layer.

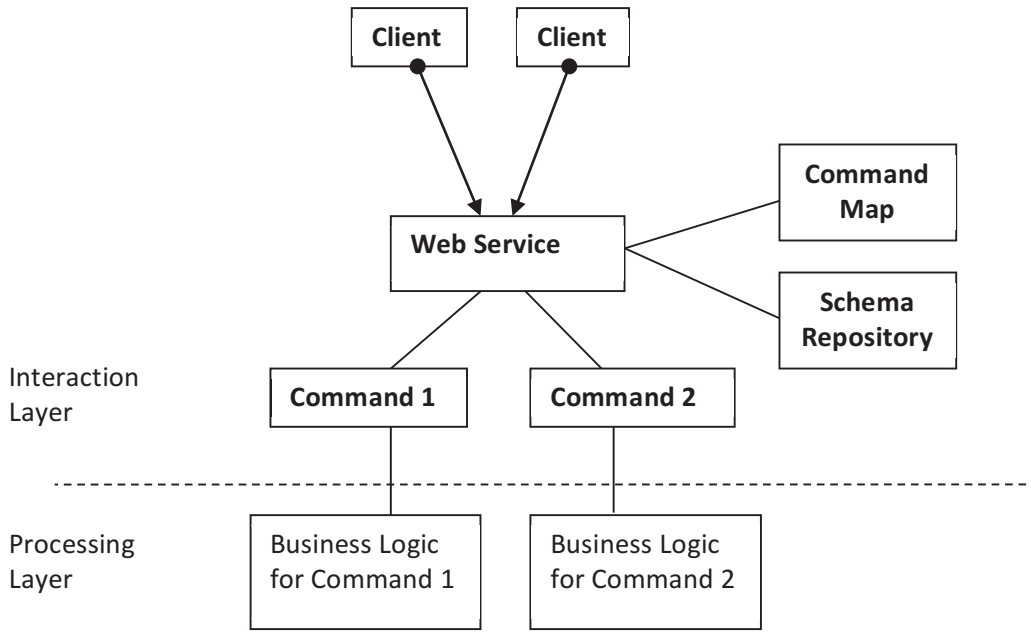


Figure 2.6: Command pattern in Web Service

Using this approach, we can define additional commands to include some of the document content only, thus increase the extensibility of the interface. This pattern can also be used as a centralised manager broker for Web Service interactions to consolidate their interaction layer. This can consolidate all clients' incoming and outgoing message interactions, and route the requests to the appropriate business operations in the processing layer. Without a centralised manager, a business enterprise may have many point-to-point interactions with internal and external partners, which results in multiple remote calls for one transaction.

According to Jakub (2006), applying the command pattern in distributed applications allows the execution of multiple calls as one bulk call, which can greatly improve performance by eliminating unnecessary remote calls. However, this Web Service Command pattern requires maintaining a list of commands or a command class along with a list of schemas in the repository. No design methodology has been mentioned regarding how to turn a Web Service into a command pattern interface. The selected operation "submitDocument" is too general, which ultimately can be applied to any document.

2.6 Discussion

In this chapter, we have reviewed several e-business document standards and their design patterns focussing on how the standard business documents are formed, their common elements library, the operation invocation styles and in particularly on the extensibility and reusability of the elements and operations. These documents all promote interoperability by using the UN/CEFACT Core Components-based elements. The use of "command" or "action" code list in the message allows flexibility and extendibility. Flexibility is due to multiple different commands being sent and processed. Extensibility is due to the Code list being added or deleted without affecting other documents. However, these approaches still contain redundant or repeated elements (buyer and seller details) and are therefore inefficient and slow.

Based on the research studies in this chapter, several implementations of standard business documents from BOD, UBL, xCBL and GS1 may be based on different Web Service interfaces design, different interaction patterns and may have different schemas; despite carrying out the same business logic (e.g. a purchase order request). For instance BOD, xCBL and GS1 patterns are integrating process indicators (processing codes) and business information (the actual message content) in the XML document. Hence these patterns represent a more robust abstraction interface where processing instructions (unique change codes) indicate how to handle the message. Unlike UBL and OTA, which contain no action or processing instruction inside the document, these interfaces are more focussed on fulfilling the requirement of the business process. Business applications may require multiple design aspects to suit their requirements. Thus after finding an appropriate service for integration, a client must study and analyse the WSDL description in order to understand the interface and data structure, then program an appropriate module to invoke the service. These studies have shown that most existing business document design patterns are based on coarse-grained design and expose the operation through the business process (e.g. request order, change order, cancel order, response order and etc.), i.e. one operation corresponds to one business process and consumes one business document (Section 2.4). For instance, is the "ChangeOrder.xsd", "CancelOrder.xsd" different to "Order.xsd"? Then there will be a requirement for three different business processes and three different business documents to fulfil the goal.

The advantages of a coarse-grained based transaction are two-fold: firstly, it has the ability to send a complete purchase order that contains multiple items in a single business document; and secondly, as a consequence of this, instigate a single transaction process. This is in contrast to multiple fine-grained invocations that process items individually. However, updating a purchase order that adopts the coarse-grained approach means sending a revised purchase order that can be processed in two ways: either by discarding all previous orders and processing with the updated document; or by finding out the differences in the updated order and processing those differences. In either case, this will incur difficulty and require unnecessary additional processing.

Based on research studies so far, the difficulty can be overcome by putting a processing instruction into the document element. By adopting the Command pattern, every object can be deployed with different actions. As a result, the client is sending clear instructions to the service provider including only the information that needs to be processed and avoiding any unnecessary data. Put in other words, the Command pattern can be utilised in a Web Service design by treating the document as an object and this document as a collection of different business objects. For example, there could be an Order.xsd, which is a collection of "order detail" objects, "customer party" objects, and "order item" objects. The remote "command" interface will become the Web Service interface. The "ConcreteCommand" interface will become the "action" switching interface in the session facade. The receiver will receive the "action" and invoke the business logic accordingly. In this way, defining one operation for each business document will permit business content for processing to be the focus, in other words now exposing the operations through the business documents instead of business events. In other words, instead of advertising a Web Service interface having these operations on this document, one can say that this Web Service interface can have any operations on this document.

CHAPTER 3

REVIEW OF EXISTING WEB SERVICE INTERFACE DESIGN, BUSINESS DOCUMENT DESIGN AND WEB SERVICE COMPOSITION DESIGN METHODOLOGIES

In this chapter, in order to study Web Service interface design we discuss several Web Service interface design methodologies (Section 3.1) through several literature reviews and especially focusing on Business Document design and Document Engineering, which are the bases for UBL, documents design (Section 3.2). Web Service composition (the orchestration sequence between multiple services) allows many e-business partners to complete a procurement process such as communicating with partners for stock enquiry services to other partners that handle stock ordering services and to partners that support the delivery booking services. In Section 3.3, Web Service composition designs will be examined including the standards and requirements in the Web Service composition to help further understand what are the elements in the services composition and how to design a composable service. This will address whether the proposed interface design method can replace any existing Web Service interfaces or be composable by other Web Services (i.e. in order to maximise its reusability). Then in Section 3.4 the current challenges of Web Service design methodology are discussed.

3.1 Existing Web Service interface design methodologies

3.1.1 Web Service design based on Elementary Business Function

Feuerlicht (2004) proposed a Web Service interface design method based on decomposing complex business documents into elementary business functions, and then

mapping these to Web Service operations. The author argues that the traditional e-business approaches that rely on the document-centric model often require a large and complex data structure, which may result in redundant and duplicated data elements. This design methodology can be utilised to transform a document-centric interface into a well-designed fine-grained interface resulting in a higher-level of interface abstraction.

The first step is to identify service operations by decomposing a complex business function into elementary (Atomic) functions, including identification of input and output parameters for each operation. This step is similar to maximising the method's cohesion, for example, in the object-oriented design. This assists the cohesion of a function since atomic functions represent a single task and cannot be decomposed any further. The second step is to refine the service interfaces, which aim to minimise interdependencies between applications and side effects. This step is similar to the minimisation of method coupling as in the object-oriented programming. The author applies data normalisation rules to assign the input and output parameters to the elementary functions. In general, rules 1 and 2 state that output parameters are determined by input parameters of that function only; the parameters will form a minimal set of input and output. These rules imply that the parameters are mutually dependent. Rule 3 defines that output parameters must be fully and functionally dependent on the input parameter set. The resulting set of output parameters must be directly generated by the set of input parameters. Applying and following these rules to refine the service interfaces can lead to maximised encapsulation and minimised method coupling, thus the removal of any redundant data.

This method helps to transform a document-centric service interface into a well-defined service-centric interface by following the design framework. The main goal of this design method is to rely on the principles of maximising method cohesion by: firstly, decomposing a large business process into smaller fine-grained service operations; and secondly, minimising method coupling by applying the data normalisation technique. The author claims that this method follows the standardised service interfaces as in the API (Application Programming Interfaces) design, and the benefits include: software

reliability, reusability, extensibility, maintainability and evolution by publishing new versions of existing interfaces. However, this service interface design method can only be applied to transform a coarse-grained business process into multiple fine-grained service interfaces. The outcome leads to more service operations for a given interface and thus increases the number of RPC calls for a business function. It is difficult to apply this method to a business document since it is a large coarse-grained message usually designed to accommodate only single business processes. For example, a purchaseOrder cannot be decomposed further into smaller fine-grained service operations.

3.1.2 WEB SERVICE DESIGN BASED ON REQUIREMENT ANALYSIS

Lau and Mylopoulos (2004) proposed a methodology for designing a Web Service based on Requirement Analysis. Software services are designed by starting with stakeholder goals, and by analysing these goals in order to define alternative solutions. The first step is a requirement analysis, which consists of two phases: early requirements and late requirements analysis.

Early requirements analysis is concerned with understanding the organisational context. During this type of analysis, developers identify the domain stakeholders and model them as social actors and identify aims to be achieved. Late requirements analysis is concerned with defining functional and non-functional requirements. Here the conceptual model developed earlier is extended to include functional and non-functional requirements. The next step will be architectural design; this will identify each actor's ability along with the actor diagram and assigned skills to each person. Therefore, data types and activities involved in achieving a system goal can be identified. Actor capability then transforms into operations to perform specific tasks along with the data types as inputs and outputs.

This design method is considered to be more generic and customisable, and the objective is to satisfy the client's requirements by using this method. Requirement analysis provides a mechanism to express additional detail regarding Web Service behaviour; alternative solutions can be determined through analysis to satisfy the goals

of major stakeholders. This method allows developers to understand the work scope of each requirement but does not take into consideration the granularity design of the services.

3.1.3 Web Service Design based on Transactional Service

This methodology proposed by Schmit (2005) is based on UML, which separates the four concerns of structure, transactions, workflow and security. This method is intended to solve the fields of distributed long running transactions. The four concerns can be modelled by an independent expert using OCL (Object Constraint Language) to establish a reference between diagrams.

The first layer is a Structural diagram and this method uses the UML state chart diagram as a modelling tool; only functional aspects are defined in this diagram. The structural diagram's elements can be referenced from higher-level diagrams using OCL. The second layer is a Transactional diagram formed by a UML class diagram. The designer can use OCL references at this layer to identify locations within the structural diagram where transactions are started, committed or aborted. The designer describes the additional constraints using a UML profile for the transactional diagram. A transaction is depicted as a Business Activity (a long running transaction), or an Atomic transaction (an ACID transaction) (OASIS, 2007). The transaction diagram includes some features (Quality of Service – QoS) such as compensation (composable) and timeout; participating in Web Services and referencing from the start until the end of that transaction. The third layer is security; the methods propose the inclusion of security parameters, for example Web Service calls/transactions needing to be encrypted or signed. The final layer is workflow, which represents the high level view of the composite Web Services. This layer will consist of reference elements of the structural and transaction view, and may reference some standard patterns typical of Web Services.

Following the modelling, a methodology can help the software designer to identify some common mistakes early in the development phase, which occurs in the following four areas: Structural, Transactions, Security and Workflow. The author claims that

using separate diagrams for different design aspects makes the model easier to view, and different experts can work on the design simultaneously. However, this method does not cater for reusability and extensibility.

3.1.4 Web Service design based on Shareable Components

Share Service Approach proposed by Radeka (2003) is a top-down Web Service design method, which defines shareable component services that can be reused in multiple enterprise applications. This method allows us to decide how to scope component services and then create them so that they can be combined and recombined at will to produce new aggregated services. To do this, multiple contexts must be understood so that a service is not too particular within the initial context to perform the task in a different context. The next step suggested by this method requires an understanding of the business process. In this step, several scenarios are written to describe the needs of users, what the users want to experience and what is likely to be most important to them. This will help to identify the touch points where services could be shared across problem domains without impacting on the schedule. These scenarios are then used to develop high-level business process models. The models are beginning with a rather low level in details and are then manipulated to maximise the number of processes that were reused in all diagrams. This will identify opportunities for shared services. As the services are defined the business process steps are replaced by the names of the services and with the information required to flow from service to service. This is the basis for the beginning of writing the interfaces. The next step in the process is to design the XML interfaces between the services.

The main objective of this method is to design shareable services that can be used in multiple contexts. By keeping the overall process general and the documentation simple rather than focusing on a single context, models are kept in sync with the reality of a changing environment. However, this method does not cater for extensibility.

3.1.5 Web Service design based on Data Centric Approach with Factual Dependency

Factual Dependency was described by Baghdadi (2005) as a dynamic constraint between two attributes, X and Y. The concept of factual dependency makes it possible to aggregate attributes that describe tangible or intangibles elements with respect to business events. An attribute Y is factually dependent on an attribute X if the attributes X and Y are concerned with the same Create, Retrieve, Update, Delete (CRUD) operation, i.e. having the same interface. First, it is necessary to identify business objects/artefacts as described in the universe of discourse; then define the attributes from the business objects. All attributes can be combined to form all factual dependencies. In order to reduce the factual dependency to only what is relevant, this method utilises the well-known functional dependency to address the relationship of attributes within a business object with reference to the CRUD operation. For example a create PurchaseOrder requires id, name, address, balance and payment. Then from these relationships, one can identify the business event corresponding to the CRUD operations. A business event is the cause of the attributes' values. Web Services can be generated from these business events along with input and output depending on the operation.

This methodology concentrates on a single business object and from this object, one can realise the Web Services from the attributes. However, due to the complexity of the XML Document in B2B interaction, there is generally more than one business object contained in the document. For instance, it could comprise a flight booking process with "Flight", "Traveller", "Payment", or business objects, which in turn generate many redundant factual dependencies for every operation.

3.1.6 Generic Web Service Interface design methodology

In order to extend the functionality of a Web Service, a new version of that service needs to be released in tandem with the old version to maintain backwards compatibility with external parties. This incurs additional development costs and time when trying to maintain multiple versions or introduce adaptors for integration compatibility. Generic

Web Service as proposed by Vadym (2009) can help add new functionality while preserving backwards compatibility. This Generic Web Service interface design method is based on the concept of Generic interface. According to Vadym (2009) Generic Web Service (GWS) is defined as a Web Service having at least one operation with a relaxed signature, which reduces operation signature rigidity. Signature rigidity encourages flexibility when defining the operation signature (input and output parameters).

This method relies on the relaxation in the input and output parameters to dynamically define the semantics of the operations at the run time rather than being statically defined at the design stage. In other words, based on a Generic Web Service interface, the signature and semantics of an operation can vary, depending on the Service calls. As a consequence of this, a generic operation will contain at least one identity or controlling parameter that defines or extends its operational semantics. This method can be utilised to transform a set of operations that perform the same kind of functionality and are replaced by a generic interface. For instance, instead of exposing several Web Services operations referring to a data set, one can replace them with a generic interface that executes the same functionality with one or more identity parameters, which retrieve the specified result sets. The advantage of GWS is that it adds functionality by extending the domain of its identity parameters without changing its operation signature. GWS also benefits from functional aggregations that can replace several fine-grained related operations with a generic relaxed operation. However, a GWS interface might or might not support all types of control identity. Invoking the interface with an incompatible identity might result in unexpected behaviour.

3.1.7 REST STYLE WEB SERVICE DESIGN METHODOLOGY

Traditional e-business Web Service designs are based on SOAP with loose coupling designs similar to document style; whereas REST (Representational State Transfer) Services use a uniform interface for the invocation call; therefore, the same interface can be applied to the different e-business domain to access the resources. However, REST is not a standard; it is an architectural style, which is similar to World Wide Web (Rodriguez, 2008). A developer can take this approach and apply it to Web Service design. This interface contains a fixed set of standardised operations; the operations in

the interface depict the basic operations needed for requesting information (the get operation), creating entities (the create operation), updating entities (the put operation) or deleting entities (the delete operation).

REST is based on URLs (Uniform Resource Locators) and four basic methods are associated with the content to execute the operation. Simple commands such as: GET, POST, PUT and DELETE will cover all cases of invocation. As stated by Fielding (2000), in REST architecture, operations are defined in the messages, and each object is identified by a URL, which supports the defined standard operations. Furthermore, each invocation will result in the transfer of a representation of this object (typically in JSON (Java Script Object Notation), XML or Plain Text). REST Web Services can be either stateful or stateless. Stateful require the server to maintain the state and control the flow of the client application while stateless run independently of the server. Clients will maintain the stage and decide the page flows by themselves. As describe by Fielding (2000), HATEOAS (Hypermedia As The Engine Of Application Stage) is a constraint of the REST architecture application. As REST is a resource-driven approach architecture, all resources (i.e. entities) besides containing the values should also include its URI in the response. For example, a purchase order will contain a list of items, and each item should contain a URI in the response to provide clear information on how to access these resources. A hyperlink of the item will provide a REST action (GET, REQUEST, PUT, DELETE) request to the item for additional information. As a result, the REST interface provider can develop it to support unlimited possible extensions to the client. A server can provide concurrent support of both new and existing interfaces through the versioned API such as /REST_SERVER/apiv1/poservice and /REST_SERVER/apiv2/poservice.

According to Costello (2005), in order to design Web Services in the REST style, you firstly need to identify all the conceptual entities that one wishes to expose as services (e.g. PurchaseOrder, parts and Customer). Secondly, it is necessary in order to create a URL for each resource locator. A URL is typically a combination of the “Address”, “Object” and a “Process ID”. A GET request from this URL will result in a XML document giving more information about this object and a POST/DELETE command is

used to manipulate the objects in the document. A URI is usually formatted by "web domain address/entity/identifier". Using this document along with the four methods can change the state of the object by navigating to a different URL. A URI within a document will enable a client to explore further information related to that entity. The REST style does not require the client to know any details about the process's implementation, but the exchange data formats need to be agreed upon by both parties. Therefore, a client interface can be reused for all types of processes as long as the data structure conforms to the server required. Application can access and consume the service by using the given URL. For example, interfaces to retrieve an Order ID "ABC123" would be "GET /REST_SERVER/poservice/ABC123". To delete an OrderID "ABC123" they would be "DELETE /REST_SERVER/poservice/ABC123". REST can provide fine-grained service such as delete an item in a purchase OrderID "ABC123"; it would be "DELETE /REST_SERVER/poservice/ABC123/item/2".

REST style helps simplify the Web Service interface and works well in the context of the Web. However, REST style relying on the four standard methods for invocation for simpler interfaces, can limit and restrict the service interface description. In contrast to SOAP standard based Web Services, which clearly provides the WSDL endpoint, the operations, the schema, the header (sender, receiver, security, encryption) and the error handling. Most of these pieces of information are neglected in the REST style thus Web Service clients and providers need to come up with some custom solutions in order to communicate successfully.

3.1.8 VALUE-BASED SERVICE MODELING AND DESIGN

Weigand (2009) proposed a service design method that starts from a value model and then identifies the core and enhancing services and possible Web Services based on the business engineering approach. Weigand (2009) defined the service as a valuable resource to be processed and exchanged between parties in a value network. A service must have a goal to modify (i.e. add value to) other resources. To realise a service, the process must achieve at least the goal of the service. Below are the three steps required in order to model and identify these services.

Step one is to model the business activities from an economic perspective using e3value modelling devised by Gordigin (2000). It helps to understand which enterprises and actors are involved in the exchange of resources. This model defines actors, resources, value ports and value interfaces. An actor is a business entity such as any party involved (e.g. client and provider). A resource or value object is the benefit value between the actors (e.g. purchase order). A value port is the channel in which the actors provide or receive the resources and the interface is the collection of ports that belong to the same actor. A value model represents the exchanges or conversions of the resource. The modelling begins by identifying the focal actor and services needed by that actor. These services are complementary services, supporting services and core services. Complementary services (Weigand, 2009) are those that are part of the same service exchange process and their goals concern the same resources. Supporting services are services that aim to produce another service. Core services are those that have the goal of being included in another service.

The second step is to identify more business services based on the value model in step one. A business service is an economic service provided by an actor to fulfil a customer's need, such as enhancing services and coordination services that support the resource exchanges. Enhancing services are services that add value to another service such as: firstly, a publication service which provides information about another service or resource; secondly, an access service that invokes another service; and thirdly, a management service which aims to maintain or optimise other services (monitoring, controlling, authorisation and evaluation). Coordination services are used in the exchange process to ensure the communicating parties in a business relationship are coordinated and synchronised. The business service identification will produce the specification of business rules and policies governing the services. The final step is to identify the software service at the information level and infrastructure level. This can be done using the top-down approach or the meet-in-the-middle approach.

As noted by the author, the service identification itself is very much a negotiation process and cannot be logically derived from the e3value model. However, generic solutions can be applied and their implementation can be based on standard business

practices. The advantage of the proposed method is the identification of services and this leads to potential Web Services, and helps to identify additional functional properties such as security and availability.

3.1.9 Model Driven Design of Web Service Operations using Web Engineering Practice

Ruiz and Pelechano (2007) proposed a Web Service design process based on the Software Process Engineering Metamodel (SPEM) developed by OMG. This design process follows three disciplines: Requirement elicitation, OO-Method (Object-oriented-method) and OOWS (Object-oriented-web-solution). Requirement elicitation creates a requirement model based on an analysis of the user's needs. The requirement model is based on the concept of task. The operations (the functional requirements of the system) can be identified from the task diagram. A task diagram is defined for each party involved, its goal and the user's activities to achieve these goals. The interaction between the users and systems are also described for each task. Additional leaf tasks are derived from the task diagram (i.e. decomposed into subtasks by following structural refinements). The candidate operations can be identified based on those leaf tasks that do not participate in a structural relationship and the parent tasks of a structural relationship.

Operation arguments are detected from the graphical descriptions associated with the task. Each node in the activity, i.e. a system action or an interaction point, depicts the achievement or goal of a task. Based on the output and together with the participant class in the operation, the arguments and its entity can be obtained. Then from the requirement model, the OO-Method defines the conceptual model of the application requirement. This step's output is a conceptual model consisting of class, state, sequence diagram and functional models. OOWS defines the web interface of the application based on the conceptual model. These include identifying the users, navigation and presentation models. User models are identified based on their assigned role, access permission and associated functions. The operations are identified based on their authentication, administration and management of user's permission. The navigation model defines the valid navigation paths for the system. Operations defined

by the exploration link and sequence link can facilitate retrieving the information based on the navigation context. Finally there is the presentation model, which comprises the operations for handling the information paging, ordering criteria and information layout.

The proposed method based on the OO-Method and the OOWS method guide the developer in the operations defined in the requirement model. After the transformation process, we can use the tool to generate the WSDL document. However, this model-driven design does not consider the granularity of the operation. Most of the time, the operations realised are too fine-grained and this results in a large number of operations being required for the application.

3.1.10 PRAGMATIC WEB SERVICE DESIGN APPROACH

Millard (2009) presented an agile method known as Service Responsibility and the Interaction Design Method (SRI-DM) to achieve a pragmatic Web Service design. The design process is an agile approach, which is iterative and adaptive to changes in requirements. This methodology includes three parts: a scenario, service profiles and a sequence diagram. It begins with a scenario that describes a problem and a model using case diagrams at a high level and a brief narrative description, which explains the roles of the different actors involved. Service profiles provide the abstract description of several services that act on the same data model.

Service Responsibility and Collaboration cards (SRCs) are used for modelling the capability of a service in the case being considered. The service name is on the top of the card. The left hand side of the card indicates the responsibilities of the service (what it is and what it does). This is also used for defining the granularity of a service. Initially, fine-grained services are realised and then subsequent iteration will form a coarser grained message. The right hand side of the card lists and groups other services that fulfil the identified responsibilities.

In order to design the SRC a six-step process is involved. The first step is to identify the verbs in the responsibilities that indicate the operations and nouns, which are the data models. In the second, it is necessary to group the responsibilities based on their

operations. Third, move common operations from other SRCs to the collaborations, and in the fourth, link the responsibilities with the collaboration. In the fifth step, it is necessary to compare the design with different scenarios and finally re-factor them if necessary after revisiting SRC and other SRC. A sequence diagram is used to represent the interactions between different services, which also verifies their service's responsibility in the previous step. The author notes that this design method only provides an overview model rather than a detailed process model. It is up to the developer to turn the SRC responsibilities into a service. All processes need to be revisited many times until the service interaction and responsibilities are well understood.

3.1.11 Evolving Web Service interface design

Kaminski (2006) discussed the version management problems of the Web Service and proposed a method based on the Chain of Adapters approach. The requirement for the proposed interface must be backwards compatibility. Existing clients must continue to function correctly when the service migrates to a new version. The application must use a common data store regardless of which version is running. This entails creating a v1 of the interface with namespace v1 from the current interface. When developing v2 of the interface, an adapter for exchanges between v1 and v2 also has to be created. First, if there is an additional parameter to an existing operation, the adapter must provide a default value for this parameter. Second, if the data element is changed, the adapter must provide the mapping for the translation. Third, if the operation is removed, then the adapter must implement this method. Fourth and last, if the contract of operation is changed, the adapter must implement this contract to compensate for the changes.

When a new operation or new optional element is added to the interface, there are no required changes for the adapter to consider. The adapter is handling for processing the differences between the new and old versions. Subsequently, if v3 of the interface is ready for release, then a new adapter has to be created for v2 and v3. Therefore, if the client is using v1 of the interface, the adapter between v1 and v2 will be processed first, followed by processing the adapter between v2 and the current interface. Any existing interfaces and adapters will be fixed or frozen and not configured again to compensate

for the effect of any downstream adapters. Thus only the new and current adapter needs to be configured when a new version of the interface is released. These will form a Chain of Adapters that will support multiple versions of the interface concurrently. Furthermore, the developer can write an adapter by merging the changes between any versions to bypass those versions in between (i.e. adapter between v1 and v3 will bypass v2).

3.2 Business Document Design

3.2.1 Web Service design based on Document Engineering

Web Service relies on message exchange to fulfil a business goal between two parties. Therefore, we need to assess current design methods for business documents that refer to Web Service interface design. One of the most common techniques is the "Document Engineering approach" which was adopted by OASIS to develop the UBL standard. This methodology proposed by Glushko (2005) employs documents as a common interface to retain a clean and stable relationship with the business partner despite some changes in internal business processes. It is concerned with the semantic components in the document being exchanged and the information exchanged within and between enterprises and the techniques for contextualising them for a domain. The author uses two dimensions of model abstraction and model granularity to define a model matrix, revealing the relationship between models of processes and models of documents. At the centre of the model matrix where processes are described as transactions, processes and documents constitute two parts of the same thing. Process descriptions emphasise business concerns and determine if ways of doing business are compatible. Document descriptions emphasise information content and also determine whether business systems are compatible.

The Document Engineering method requires a set of analysis, assembly and implementation tasks, and the first step is to understand the context of use. Developers are required to identify the requirements and the rules. They must start from a high level in order to understand the main business activity, party and organisation involved. This task is done by drawing a business use case diagram. The model at this level describes

the broad context of how documents and processes are utilised. The second step is to allocate patterns for the Business Process. Patterns are models that are sufficiently general, adaptable and reusable; Document Engineering emphasises the reuse of existing specifications or standards. Doing so will reduce costs and risks while increasing reliability and interoperability. Patterns may be structural, presentational or content patterns and this is realised by drawing activity diagrams. Within the activity diagram, operations are identified by any interaction between two parties. The next step is analysing documents and their components, and after we have realised the operation or business process, the selected process model or pattern will identify the roles that a document plays. Analyses of documents will reveal the business rules that govern the content, structure, presentation, syntax and semantics of the information contained in them (Data Style Sheet). The component analysis phase starts with the harvesting task. This identifies the individual semantic components contained in each selected document. In the component assembly phase, we assemble sets of these information components into meaningful structures to create a coherent conceptual view known as the document component model (XSD Schema), using data analysis techniques that normalise the components into a structure based on their functional dependency. The fourth step is designing the Document Model; we create models for new types of documents based on the components, structures and associations that satisfy the context of use (Document Assembly Model or XML, for instance). Using functional dependencies to normalise the XML document for designing the logical structure, this ensures that all data elements in a group are discrete and all members apart from the primary identifier are functionally independent of one another. The last step will implement the Model; model-based applications can then be initiated using software whose generic functionality is made context-specific. This is done by configuring or extending it to use the context dependent information and behaviour specified in the model.

With reference to service reusability, the Business Information document must be built using common building blocks or Core Components. A Core Component is defined as a building block that contains pieces of business information belonging to a single concept, for example a person. In order to model the class diagram for business

information exchange, we must determine the business entities that are affected by the transaction (e.g. Security Person). Each business entity is described by the information needed to fulfil the transaction only. The information is built by reusing core components and putting them into the context of the business transaction. This will become the Business Information Entity (BIE). Eventually, a business document will contain many of these BIEs.

The main purpose of Glushko's method is to focus on analysing and designing documents that accommodate both partners' requirements. These documents are designed to be generic so they can be reused in different business domains, for instance a Purchase Order. However, we can apply these documents to any e-business application with a specific context (for example, Purchase Order for Furniture). This method is approached by reaching a common understanding about how their processes should be designed, how they can be decomposed into document-based service components, and the information they exchange with the documents. However, the author did not specify how to identify the input and output of an operation, nor suggest how to define the granularity of these operations. Although this approach is widely used in the industry, the resulting documents are complex and contain redundant elements (e.g. CancelPurchaseOrder needs to be sent to all parties with purchase order details). We can extend this to our methodology by adopting these documents and re-engineering them with the focus on Web Service interface design. This includes, for example, identifying the atomic operation in the document and adjusting the granularity for the best result.

3.2.2 Web Service design based on UN/CEFACT's Modelling Methodology (UMM)

An alternative to the Document Engineering method is UMM, which also uses Core Component to build up business documents. This method proposed by Christian (2005) was the first design for ebXML but was later extended to Web Service design. UMM concentrates on the business semantics of a B2B partnership; the goal of UMM is to capture the commitments made by business partners. UMM consists of four views or layers. The first layer is the Business Domain View (BDV), which is used to gather

existing knowledge. It defines business processes in the domain of business problems and is important to stakeholders, while business processes in this stage are discovered but not constructed. The second layer is the Business Requirement View (BRV), which identifies possible business collaborations and harmonises the requirements of different stakeholders. This outcome of a requirement's specification for a business collaboration is shared by stakeholders. From the BDV, we can identify the multiparty collaboration, and we then decompose it into a binary collaboration (between 2 parties only). From this binary collaboration, we can decompose it into a business transaction. Business collaboration is designed by choreography of business state change, which represents the lifecycle of the business collaboration.

The final layer is the Business Transaction View (BTV), the objective of which is to ensure that all the identified services realise all the required business processes. This transforms the requirements into an analysis model; the analysis model defines the business collaboration protocol, which defines the choreography amongst the business transaction activities. The choreography (activity graph) is always composed of two business actions and the data model (class diagram), which represent the document of information exchanges. Business information is described by views on reusable core components. This is followed by defined service flows that depict the sequential inter-component service interaction of a business process, and a defined service composition depicting the view of the service orchestration. It is expressed as BPEL and WSDL. The last step is to define the inputs and outputs of these Web Service operations by mapping the UMM model to XML schemas. There is an optional layer Business Service View (BSV) that transforms the artefacts of the previous view to show message exchanges between network components.

This method focuses on analysing and identifying the collaboration between partners. Through this collaboration, we can define the required business transactions as well as business documents. This approach tends to be coarse-grained because documents are defined based on business activity but not on operations.

3.3 Existing Web Service Composition design methodology

Web Service compositions aim to fulfil the requirements of a standard-based coordinated and collaborative service that support many stakeholders and service transactions (Hall, 2003). This is an important aspect of making Web Services reusable, and providing a closer representation of business transactions across the application domain. Especially in the Global Trade Domain where a purchase order can be sourced from multiple different vendors (i.e. the need to fulfil an order from different Web Service providers) and the dispatch might be requiring to contact multiple delivery Web Service providers for the service. As a result, the study and research of the current Web Service composition design methodology will allow us to understand whether the propose interface design method can replace any existing Web Service interfaces or be composed by other Web Services. One language that aims to consolidate previous efforts by specifying a composition language is the Business Process Execution Language (BPEL). BPEL is mostly the result of work undertaken previously by industry standards to build such specifications, such as from XLANG (Microsoft) and Web Service Flow Language (WSFL) by IBM. BPEL is recognised as a standard service composition language by OASIS and has been positioned in a standards stack for clarification, with the consensus of where it is related to other emerging standards. BPEL is a specification language that expresses the composition of existing Web Services. Expressions include how to invoke the external Web Services, and the execution order of operations and elements. The BPEL models the composition services as a normal service and can be published for external invocation.

3.3.1 WEB SERVICE COMPOSITION

BPEL combines the efforts of both Web Service Flow Language (WSFL) and XLANG. BPEL is a XML language for Web Service composition jointly developed by BEA, IBM, Microsoft, SAP and Siebel. BPEL was approved by OASIS as a standard (Vander, 2003). In BPEL, the composition outcome is called a process and participating services are called partners. Message exchange or intermediate result passing is referred to as an activity. BPEL offers the ability to scope activities and specifies fault handlers and compensation handlers for scopes. A fault handler is executed when an exception arises

while compensation handlers are triggered due to faults or through activities that force compensation within a scope.

The requirements for Service Composition are as follows:

i. **Connectivity**: this requirement describes the possibility of reasoning about input and output parameters of a Web Service. Milanovic and Malek (2004) claim that currently, all composition approaches offer service connectivity. Different approaches have various ways of modelling the connectivity. However, they aim to map and orchestrate input and output messages between the partner services' ports. Therefore, with reliable connectivity we can identify which services are composed.

ii. **Non-functional properties**: service composition for non-functional properties should also be considered such as performance, execution time, cost and security. The author notes that most approaches neglect specification of non-functional properties such as security, dependability or performance. Currently, only OWLS (Ontology Web Service Language) lets users define some non-functional properties. OWLS enables automatic service discovery, invocation, composition, interoperation and execution monitoring.

iii. **Correctness**: verification of service composition is required to confirm correct executions and prevent deadlocks or livelocks. To support correctness in composition, developers have to translate the business processes into a formal language such as Pi-calculus, Petri nets or Finite state machine. BPEL and OWLS only deal with implementation rather than specification, and hence they cannot verify the level of correctness. Due to some complex systems, in reality the numbers of inputs, outputs and states are too large; we cannot apply the model to the whole scenario because translating from WSDL to mathematical solutions is time-consuming and impractical.

iv. **Scalability**: since Web Service is a distributed environment, where changes can occur regularly, thus this is required to be scalable. The approach should

consider scalability to additional functional and non-functional requirements for a service composition.

Currently, static Web Service compositions are the most used format in both industry and academia (Zein and Kermarrec, 2006). They are formed by manually identifying (i.e. by human assessment) the applicability of a Web Service to a particular problem domain. The composition is therefore limited to the Web Service encompassed in the design. Static compositions are represented by known paths; known data representations and expected results are part of a formal and technical link with the Web Service. Dynamic Web Service compositions form the basis for discovery and flexibility in Web Service invocations. Although WSDL details the technical interface and locates a given service, it does not identify what the service does, what function it performs in order to fulfil the request, and nor does it suggest what level of service it will provide.

3.3.2 Composition design based on UML

According to Gardner (2003) the author describes an approach to specifying business processes through a subset of the UML profiles (driven by a process class with attributes and methods). The behaviour of the interacting process classes is shown using an activity graph. While this approach indicates partnered processes working together, it is unclear how multiple scenarios of each process would be specified (Woodman et al., 2004). However, the authors provide examples in UML Sequence Diagrams and Activity graphs, building requirements in process algebra using pi-Calculus to represent the concurrent and alternative paths possible in a composite Web Service process. Additional work, combining model-driven-based approaches such as UML with processes specified in algebra, have been discussed by Pistore et al (2004). The authors use an extended version of the TROPOS methodology to capture business requirement and then generate a BPEL source code from these requirements. The main concern with this approach is that it appears to represent a single scenario process only.

Another composition method using UML was proposed by Skogen (2004), and this employs UML as a tool to model Web Service composition and then translate the UML model into executable WSDL specification. Firstly, we create a preliminary model of

the new composite service and identify their candidate Web Services. This step obtains an overview of the composite Web Services and identifies which services are involved. The preliminary interface will be described in the UML diagram, and the UML activity model will suggest how the new operation should be composed. We can then search for the candidate Web Services from the UDDI registry based on the preliminary model for the matching service criterion. The selected Web Service interface will then be transformed into UML to identify the actual operation. Activity flow and data flow are then modelled and transformations are necessary when the data between Web Services do not match. This step will be repeatedly designed until a final composition is produced. Then the UML model will be transformed into an executable specification (XML) and translated to the WSDL interface for publication as a new Web Service.

This method relies on utilising UML as a common integration platform. The UML composite model can produce a composite Web Service. The author also claims that the UML is a sufficient model since they can satisfy the five basic patterns for composition, i.e. sequence, parallel split, synchronisation, exclusive choice and simple merge.

3.3.3 Composition Design based on Formal Language

Interface formalisms proposed by Dirk (2005) are used to avoid errors in component-based system design. Applying this methodology, the developer can check the compatibility and substitutability of two or more Web Services. The compatibility checking algorithm verifies that two or more interfaces fulfil each other's constraints, whereas the substitutive checking algorithm confirms that a service demands fewer/higher and fulfils more/less constraints than another service. If two or more interfaces are compatible, corresponding components work together properly at the run-time, i.e. the number and types of the parameters of a function call and the function definition match. Since a Web Service often depends on another Web Service and is implemented by different vendors, using formal language to specify rules between interactions can help to check interface compatibility and refinement. For example, model checking can reduce the burden on testing for system integration and validation. Interface checking stands a much better chance of succeeding in practice than implementation checking, as interfaces are usually less complex than the corresponding

implementations. The author suggests that a good interface design should reveal all information about a Web Service that is needed to use the service properly, and that the interface does not reveal more than that.

3.3.4 Composition Design based on Case-Based Reasoning

Limthanmaphon (2003) proposed a Case-Based Reasoning (CBR) approach for Web Service composition. This method aims to solve problems by comparing them to old ones that were overcome (Leake, 1996). The first step is to identify the service case base, which is a collection of service cases. A service case is a pre-assembly composite service. The following step is to retrieve the service case from the client query; each query is extracted to match the case base in order to find the relationship solution for the composition service. Together with the constraint and relationship, the service's composer then integrates all the allocated services to achieve the intended outcome.

The case-based reasoning technique for Web Service composition can be applied in the process of service discovery. CBR are the common query results of any two or more services with the constraint limited. The author claims that there are three main advantages. First, this method can reduce the cost of composition. Second, these collaboration services are designed to satisfy the client. Third, it is an efficient service discovery. Developers need to identify all the possible cases to meet a number of constraints. The author focuses on describing the relationship between services. However, he does not define the elements in the service in such a way that they can be fitted together.

3.3.5 Composition Design based on RosettaNet PIPs

Khalaf (2006) proposed a "Template-specialization-implementation" approach for designing BPEL, based on RosettaNet PIPs. The RosettaNet standard consists of a set of specifications such as a dictionary, an implementation framework and a Partner Interface Process (PIP). The first step is to create a template (abstract BPEL) to capture the message flow pattern of the two parties. This template will show all the conversations and variables except elements/attributes. The next step is to create a fully

valid abstract BPEL process from these templates that represents a specialisation of that pattern. The abstract process will include all partner links, correlation sets, operation names and variable details. The last step is implementation, which means following on from the abstract BPEL process to define the process that is customer-dependent, i.e. different fault handler, different assigned variable, etc. The transformation starts by mapping partner roles in PIP to <Partner> in BPEL. Correlation sets, operation names and variable details are also translated and implemented as part of the executable business process in BPEL.

This bottom-up three-level approach focuses on BPEL designs based on RosettaNet PIPs. The author claims this method can be generalised to similar environments having multi-party processes. The advantages include: compatibility can be checked during the template creation step when using Petri Net or other formal methods; and compliance is evident between the abstract and executable processes.

3.4 Discussion

We have discussed several Web Service interface designs, along with various Web Service composition design techniques. And highlighted the importance of identifying the optimal level of service granularity. While some developers prefer a coarse-grained approach to minimise Web Service invocation, others prefer a fine-grained strategy to maximise reusability. Therefore, we need to investigate the correct granularity a service should reveal for the best results. This is also one of the main objectives of this research.

Looking at fine-grained Web Service design Feuerlicht (2004) investigated Web Service designs with an approach based on decomposing and mapping atomic functions to Web Service operations. These are used to transform any document-centric interface into a fine-grained service interface. The author focuses on minimising data coupling so that inter-dependencies and side effects are minimised. This approach can be used for any web application design within a vertical domain. On the other hand, Radeka (2003) uses business process decomposition to identify Web Services. This involved creating a

scenario for each business process to identify a shared service. However, the method proposed does not address how to identify the service operation, nor suggest the granularity of a Web Service interface.

The majority of industry developers adopt the document exchange style (UBL, UN/Cefact, OTA) (Glusko, 2002) due to e-business requiring loose couple operations. However, this approach as pointed out earlier will contain excessive amounts of data because these documents contain repeated and redundant elements, i.e. they are self-described. Furthermore, making changes to and updating any record will eventually require sending an updated version of that document to accommodate changes. This approach is inefficient. Since a coarse-grained Web Service contains large document schema, to simplify the Web Service message (i.e. XML schema), Provost (2002) describes using the normalising method in relational database design to eliminate ambiguity, minimise redundancy and facilitate preservation. However, applying this method on a large scale can be complex and time-consuming.

If all enterprises develop and publish their Web Services only according to their own business agreements, then enterprises will have programming interfaces that differ from those of other enterprises even if they conduct the same business. Thus after finding an appropriate service from the UDDI registry, customers must study and analyse the WSDL description, which is issued by this publisher. They must also understand the program interface and data structure, then program an appropriate module to invoke this Service and treat the response. When switching to the partner's Web Services, programmers eventually will have to develop new modules to bind to the new service interface. Because two individual Web Services are in different data structures and operation interfaces even though they realise the same business logic, all of this hampers the adoption of Web Services and flexibility of e-business applications. Though some new tools can be used to generate the output invocation code from the WSDL document these tools' ability is still limited because they do not understand the interface and parameters precisely without human intervention.

Furthermore, there is much data to be exchanged between e-business partners. These data such as price, order, specifications, etc., are complex with special data structures that differ among enterprises. Therefore, partners have to provide many Web Services to exchange these data in XML format via the SOAP message. However, as long as every enterprise uses a different data structure to describe its data, for example adopted different XSD (XML schema definition), for the order document, customers have to program a special module according to special XSD to treat business data exactly. Consequently these different data structures form a bottleneck in the composition of Web Services.

To maximise the reusability of Web Services, we look at the design context for composition. Currently, BPEL is the worldwide standard for business process composition. Static Web Service compositions described by Zein (2006) are the most commonly used, and these compositions are represented by known path and data structures while dynamic composition forms the basis for discovery and flexibility in Web Service invocations. Other approaches such as Gardner (2003) specify business processes and describe the behaviour of the interacting processes using an activity graph. However, it is unclear how multiple scenarios in each process will be specified. Conversely, Woodman (2004) suggests building requirements in a process algebra using Pi Calculus to represent the concurrent and alternative paths possible in a composite Web Service process.

Another composition method proposed by Brogi (2004) is to map all actions and data parameters between services in choreography. This approach is used for checking consistency and preventing deadlocks occurring. When employing formal methods for compositions, while it is true that we can validate the correctness, it is impractical, time-consuming and unsuitable from the business perspective. They cannot be applied to a large-scale enterprise system. Case Base Reasoning proposed by Limthanmaphon (2003) is designed to satisfy the client's requirements. However, developers need to identify all the possible cases to meet a number of constraints. For instance, if the composition is involved with a large number of services, this will become very complex. A simpler method approach by Skogen (2004) is to use UML as a tool to

model the composition interaction and translate this into a Web Service interface. Consequently, many translation steps are required until the final composition is achieved.

Interface formalism proposed by Dirk (2005) on the other hand, focuses on checking compatibility and substitutability of any given service using the formal algorithm. This interface checking method can reduce the burden on system integration validation and testing. Khalaf (2006) proposed a "Template-specialization-implementation" composition approach based on RosettaNet PIPs. This method can be generalised to a similar environment with multi-party processes, and compatibility can be checked using the formal method.

If the composition of Web Services only includes the applications within an enterprise or only covers a limited number of partners, problems regarding composition mentioned previously may not appear to be as important. However, using the example of a dynamic supply chain in e-business, a loose coupling is necessary because this kind of supply chain decomposes or merges frequently. If every change in the supply chain gives rise to a modification or update the cost of the change is too expensive. Therefore, the e-business's information platform is inefficient. To compose Web Services quickly and cost-effectively, B2B collaboration is a solution based on the standard service interface. A common business data structure is therefore required.

There are many methodologies for Web Service, interface and composition design. Out of all the design approaches discussed so far, there seems to be no methodology that determines the right level of granularity a service interface should reveal. This is an important factor because an interface with the right granularity will increase the reusability and extendibility of the Web Services.

In this thesis, we will address the Web Service interface design problem by redefining the standard business document based on the minimalist service interface. We will propose a methodology that defines a service interface, which will provide guiding principles that help identify a consistent set of service interfaces. These in turn can be

used to compose a complex enterprise service maximising reuse scenarios, avoid duplication, and make extendibility and flexibility in a service possible. The outcome is that such a composition of Web Services can be optimised. We realised that if we design our Web Service interface so that it is reusable and optimal in the first place, then the composition of a Web Service will be easier to integrate and more practical for e-business applications. We will discuss the issues and limitations of the current methodologies in more detail in Chapter 4.

CHAPTER 4
PROPOSED METHODOLOGY

The proposed design methodology is based on the single stateless transaction mechanism and focuses on the individual message design. For multiple messages correlation design between document exchanges such as using BPEL composition will be included as part of the future work described in Section 7.3. We will describe the concept of the minimalist approach to service interface design and transforming the business document based on the minimalism concept (Section 4.1). A methodology will be proposed followed by illustrating the case study using the proposed design method (Section 4.2). At the end of the chapter (Section 4.3), we will discuss the advantages and trade-offs of the proposed method compared to the traditional approach.

4.1 Objective

Traditional Web Service designs such as transforming existing applications and exposing the method based on the Web Service interface does not provide an ideal way for Web Services reuse and extension (Feuerlicht, 2006). The proposed design methodology focuses on reusability, extensibility, flexibility and addresses efficiency concerning a set of Web Service interfaces. It does this by exposing the operation interfaces that are based on business documents rather than business events. For instance, if one is treating each business document as a business document object, then every document can be deleted, updated and contain the response outcome. This can minimise the number of interfaces, thus leading to easier maintenance (only one single document is required to handle all related business processes). Now we can externalise a smaller number of interfaces while providing similar or more aspects of the original functionality. We addressed the issue of reusability based on generic operation, other

elements and multi-grained operation (both fine-grained and coarse-grained in the message). Extensibility is addressed by the notion of the Command pattern interface, which makes it possible to add or remove commands dynamically. The interface is more flexible due to one or more commands being put into the message. Finally, efficiency is addressed by minimalism, which only exposes or sends the required data.

4.1.1 MINIMALIST INTERFACE DESIGN

Minimalism has been defined as reducing the concept or idea to its simplest form. According to McManus (2005), minimalism helps to reduce the concern for the redundant message and focuses on what is important. The idea behind the minimal interface is to design an API that allows the client to do everything they need to do, but to reduce the capabilities to the smallest reasonable set of methods. Therefore, instead of exposing several business processes for executing a business document, the developer can reveal the business document as an interface. It will provide the business actions through the document itself, thus reducing the number of interfaces required in the business process while increasing the document's control flexibility and extendibility. A Web Service with a large number of operations will not likely be used effectively. By keeping a small and focused set of methods, this will make it easier for clients to find out what the document is and what it can do.

4.1.2 METHODOLOGY

The UBL business documents have been designed to an industry specific standard; there is no requirement to redefine these UBL elements. The existing elements and their structure can be used to redefine the business documents with the proposed design method. The objective is to increase reusability, flexibility and minimise the number of interfaces by assigning "Processing Instruction" into the available elements within the business document. This is covered in the following three-step design discussed below.

Step 1: Identifying Action Type

In order to reduce the number of interfaces, this method will put the processing instruction into the document and present it as an element. For every industry standard

business document, it is necessary to first identify the actions supported for each document as well as elements in it. Business actions are verbs that are associated with the business event, and can be a simple "CRUD" action, i.e. Create, Retrieve, Update and Delete, or "Transfer", "Response", "Activate" and "Synchronise". There is no restriction on the number of actions a document can use. Then for every element in the business document, the developer will identify the "Action" type against each element, i.e. this will determine which element will support the action. Furthermore, the developer can define multiple actions as part of an element.

Step 2: Identifying Web Service operations within a Web Service interface

Step 1 will produce a matrix of "Actions" against "Elements" within a business document, which means the document can now support multiple business events. Therefore, instead of exposing several operations to individual events, the developer only exposes one operation to the "Invoke" call, and one operation to the "Response" call. When using a generic verb, i.e. "Process" or "Response", this is followed by the document name as an operation name to cover all the identified business events relating to that document.

Step 3: Defining the XSD Schema

Since there are multiple action types in one document, the "Choice" structure schema can be used to integrate all action-specific elements. The root element, however, only supports one action at a time (i.e. depending on which "Action" is being undertaken) and the developer only uses one specific element from the "Choice". Then for every sub element that supports the action, the developer will add an attribute "Action" as a processing instruction. Depending on the sub element, there could be more than one action that is supported in the same message.

4.1.3 CASE STUDY

Since this thesis focuses on Web Service interface-based application, the case study will use "Fulfilment" business documents from UBL to demonstrate current design limitations. UBL is the product of an international effort to define a royalty-free library of standard electronic XML business documents. This standard was developed based on

Document Engineering (Glusko, 2005) and uses Core Component to generate data elements. Currently, UBL version 2 is the latest version of this product.

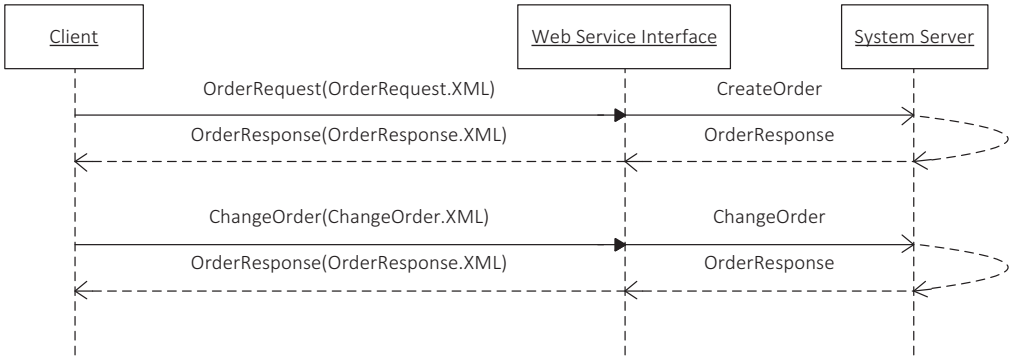


Figure 4.1 Traditional UBL Ordering Service Interface

This scenario (Figure 4.1) indicates that a Buyer Party can place an "Order" document and wait for a response from the Seller Party, which can be "Accept" or "Reject". Then the Buyer Party can change the "Order" and wait for a response from the Seller Party again.

There are five different messages corresponding to five different operation interfaces to fulfil an "Order" Business Process.

Listing 4.1 Traditional UBL OrderRequest Schema

```
<xs:element name="Order">
  <xs:complexType>
    <xs:sequence>
          <xs:element name="ID" type="xs:string" />
          <xs:element name="IssueDate" type="xs:date" />
          <xs:element name="BuyerCustomerParty" type="CustomerParty" />
          <xs:element name="SellerCustomerParty" type="CustomerParty" />
          <xs:element name="OrderDetail" type="OrderLine" />
    </xs:sequence>
  </xs:complexType>
 </xs:element>
```

Listing 4.2 Traditional UBL OrderChange Schema

```
<xs:element name="OrderChange">
  <xs:complexType>
   <xs:sequence>
    <xs:element name="OrderReference" type="xs:string" />
        <xs:element name="IssueDate" type="xs:date" />
        <xs:element name="SequenceNumberID" type="xs:string" />
        <xs:element name="BuyerCustomerParty" type="CustomerParty" />
        <xs:element name="SellerCustomerParty" type="CustomerParty" />
        <xs:element name="OrderDetail" type="OrderLine" />
   </xs:sequence>
  </xs:complexType>
 </xs:element>
```

Listing 4.3 Traditional UBL OrderCancel Schema

```
<xs:element name="OrderCancel">
  <xs:complexType>
   <xs:sequence>
        <xs:element name="ID" type="xs:string" />
        <xs:element name="OrderReference" type="xs:string" />
        <xs:element name="IssueDate" type="xs:date" />
        <xs:element name=" CancellationNote" type="xs:string" />
        <xs:element name="BuyerCustomerParty" type="CustomerParty" />
        <xs:element name="SellerCustomerParty" type="CustomerParty" />
   </xs:sequence>
  </xs:complexType>
 </xs:element>
```

Listing 4.4 Traditional UBL OrderResponseSimple Schema

```
<xs:element name="OrderResponseSimple">
  <xs:complexType>
   <xs:sequence>
        <xs:element name="ID" type="xs:string" />
        <xs:element name="OrderReference" type="xs:string" />
        <xs:element name="IssueDate" type="xs:date" />
        <xs:element name="AcceptedIndicator" type="xs:string" />
        <xs:element name="BuyerCustomerParty" type="CustomerParty" />
        <xs:element name="SellerCustomerParty" type="CustomerParty" />
   </xs:sequence>
  </xs:complexType>
 </xs:element>
```

Listing 4.5 Traditional UBL OrderResponse Schema

```
<xs:element name="OrderResponse">
  <xs:complexType>
   <xs:sequence>
        <xs:element name="ID" type="xs:string" />
        <xs:element name="OrderReference" type="xs:string" />
        <xs:element name="IssueDate" type="xs:date" />
        <xs:element name="BuyerCustomerParty" type="CustomerParty" />
        <xs:element name="SellerCustomerParty" type="CustomerParty" />
        <xs:element name="OrderDetail" type="OrderLine" />
   </xs:sequence>
  </xs:complexType>
 </xs:element>
```

Because of the nature of business documents, every business process will contain a corresponding document. For instance, if the client needs to change an Order item inside an order, ultimately, the client will need to send an updated version of the OrderChange.XML document, which contains everything including the original Order document with the change. This approach is ineffective because the transmitted document is large, complex, and redundant in that it contains many optional and repeated elements. Developing these Web Services is time-consuming and the standards organisation must make considerable effort to define the standard business document. This approach is inefficient and hard to maintain. Furthermore, if the business document is upgraded to a new version, all the related Web Services eventually will have to be re-developed. Consequently, composition between services will become a difficult task.

4.2 Design with proposed solution

Step 1: Identifying Action Type

In this case using UBL, the Order document support actions are as follows: New, Change, Cancel and Response. To make things simple the case study only considers the core elements in the document, i.e. those that are not optional.

Order.xsd	OrderChange.xsd	OrderCancel.xsd	OrderResponse-Simple.xsd	OrderResponse.xsd
	Sequence-NumberID	ID	ID	ID
ID	Order-Reference	Order-Reference	Order-Reference	Order-Reference
IssueDate	IssueDate	IssueDate	IssueDate	IssueDate
BuyerCustomer-Party	BuyerCustomer-Party	BuyerCustomer-Party	BuyerCustomer-Party	BuyerCustomer-Party
SellerSupplier-Party	SellerSupplier-Party	SellerSupplier-Party	SellerSupplier-Party	SellerSupplierParty
OrderLine	OrderLine	CancellationNote	Accepted-Indicator	OrderLine

Table 4.1 Example core element schemas from UBL

	Action Type			
Element	**New**	**Update**	**Delete**	**Response**
*Order (root)	X	X	X	X
ID	X			
IssueDate	X			
BuyerCustomerParty	X			
SellerSupplierParty	X			
OrderLine	X	X	X	
AcceptedIndicator				X

Table 4.2 Action Type for Order document with required elements

From Table 4.2, the developer can create a "New" order using all required elements; this business document also supports "Update", "Delete" and "Response" Order through selected elements. In particular, only the sub element "OrderLine" supports the "Update" and "Delete" actions. This means a client can perform "Update" and "Delete" on any specific "OrderLine". The element "AcceptedIndicator" is supported only in the action marked "Response".

Step 2: Identifying Web Service operations within a Web Service interface

Now that the method treating all actions support are in one business document, i.e. "Order.XML", the developer has to define the operations based on "Request and Response" structure, where "Create", "Change", and "Cancel" Order are defined as "ProcessOrder". This only exposes one operation instead of three separate ones. The second operation is "ResponseOrder".

Web Service Interface	Business Function	Web Service Operation	Business Document
PurchaseOrder.wsdl	CreateOrder()	ProcessOrder()	Order.XML
	ChangeOrder()		
	CancelOrder()		
	ResponseOrder()	ResponseOrder()	

Table 4.3 Summary of the PurchaseOrder Web Service interface

Step 3: Defining the XSD Schema

The developer used "Choice" structure schema to integrate all action-specific elements, which only support one action at a time, i.e. depending on which "Action", use only one specific element from the "Choice". In the case of the Cancel Order, the client only needs "ID" and "Action" elements. Now the method can incorporate all three of the schemas into one schema. The operation invoked will depend on the action type (New, Update, Delete and Response). Since all elements are designed for a specific task, clients only send the necessary data between the Services.

Listing 4.6 Proposed Order.xsd Schema

```
<xs:element name="Order">
<xs:complexType>
 <xs:choice>
 <xs:element minOccurs="0" name="NewOrder" type="NewOrderType" />
 <xs:element minOccurs="0" name="ChangeOrder" type="ChangeOrderType" />
 <xs:element minOccurs="0" name="ResponseOrder" type="ResponseOrderType" />
 </xs:choice>
    <xs:attribute name="ID" type=" xs:string " use="required" />
    <xs:attribute name="Action" type="ActionType" use="required" />
</xs:complexType>
</xs:element>
```

Listing 4.6 is the top level of the order schema, and this approach relies on the common interface (i.e. ProcessOrder ()) to process multiple business events that correspond to a business document. Details of the sub elements are reviewed in Figure 4.2. The proposed Order schema now contains: new order, change order, cancel order and response order elements. Therefore, the client only needs to send the required elements

for a business event. Messages are processed corresponding to the “Action” element, which will increase flexibility and reduce redundancy.

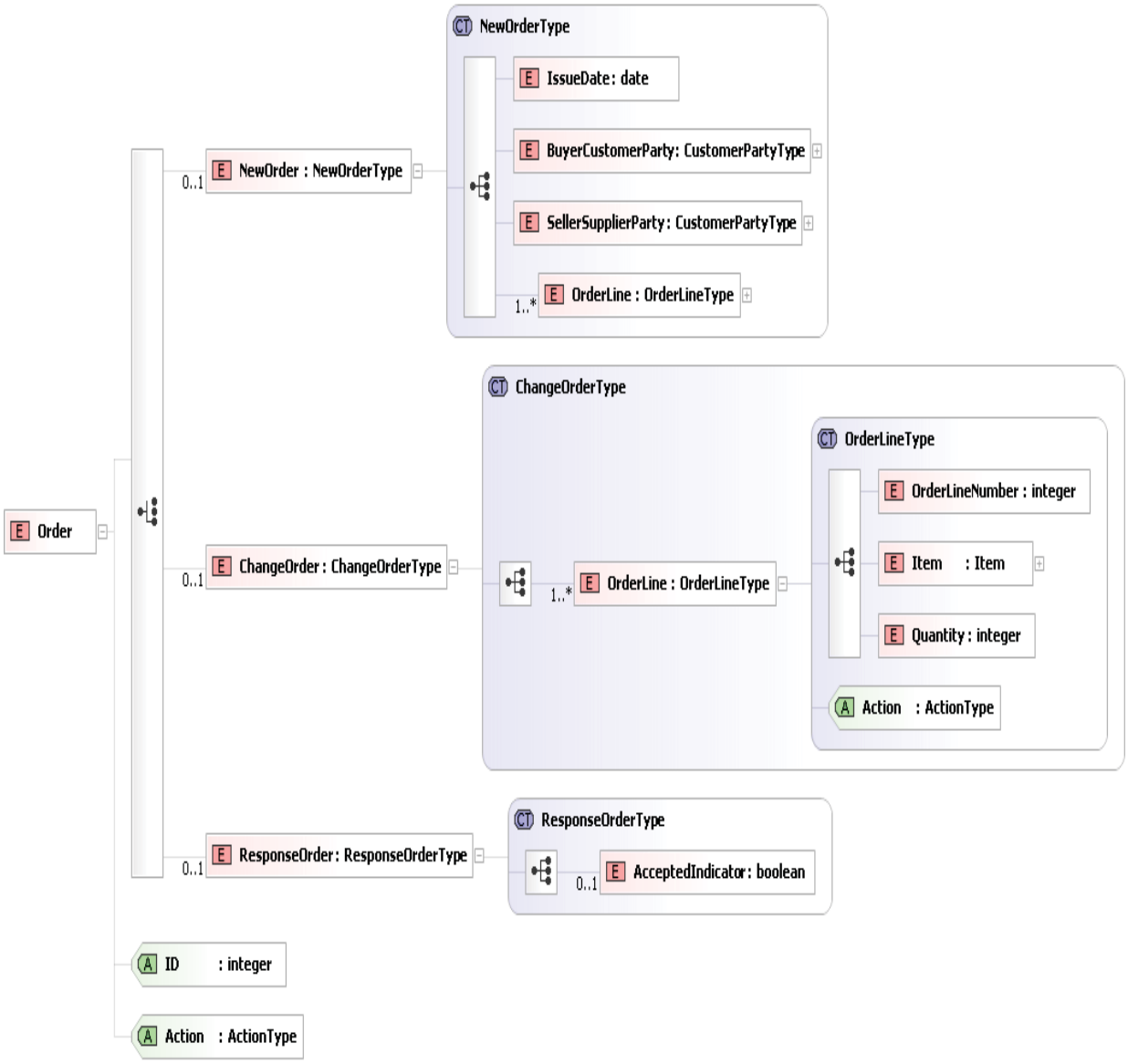


Figure 4.2 Proposed Order.xsd Schema

Sending a new Order business document (Listing 4.7) is the same as the traditional UBL Order document except it now includes the element “Action”.

Listing 4.7 Generated new order “Order.XML” with proposed method

```
<order ID="0001" Action="New">
    <NewOrder>
     <IssueDate>2008-12-01</IssueDate>
     <BuyerCustomerParty>
            <Name>Name1</Name>
            <Address>Address1</Address>
            <Contact>Contact1</Contact>
     </BuyerCustomerParty>
     <SellerSupplierParty>
            <Name>Name2</Name>
            <Address>Address2</Address>
            <Contact>Contact2</Contact>
     </SellerSupplierParty>
     <OrderLine Action="New">
      <OrderLineNumber>01</OrderLineNumber>
      <Item>
            <ItemID>001</ItemID>
            <Description>Book</Description>
            <Price>99.99</Price>
      </Item>
      <Quantity>1</Quantity>
     </OrderLine>
    </NewOrder>
</order>
```

To send an "OrderChange.XML" to execute the change and cancel an OrderLine from an Order at the same time, this XML will become Listing 4.8 which is shown below:

Listing 4.8 Generated Order.XML support multiple actions

```
<order ID="0001" Action="Update">
    <ChangeOrder>
        <OrderLine Action="Update">
          <OrderLineNumber>01</OrderLineNumber>
          <Item>
              <ItemID>002</ItemID>
              <Description>Pen</Description>
              <Price>9.99</Price>
          </Item>
          <Quantity>1</Quantity>
        </OrderLine>
        <OrderLine Action="Cancel">
          <OrderLineNumber>02</OrderLineNumber>
          <Item>
              <ItemID>001</ItemID>
               <Description>Book</Description>
               <Price>99.99</Price>
          </Item>
          <Quantity>1</Quantity>
        </OrderLine>
    </ChangeOrder>
</order>
```

The message is now more flexible to work with and consequently; there is minimal requirement for resubmitting the XML data. To send an order change (Listing 4.9) to change an OrderLine from the above Order, note that now it is necessary only to send the updated information instead of the complete XML document again

Listing 4.9 Generated update order "Order.XML" with proposed method

```
<order ID="0001" Action="Update">
    <ChangeOrder>
        <OrderLine Action="Update">
          <OrderLineNumber>01</OrderLineNumber>
          <Item>
              <ItemID>002</ItemID>
              <Description>Pen</Description>
              <Price>9.99</Price>
          </Item>
          <Quantity>1</Quantity>
        </OrderLine>
    </ChangeOrder>
</order>
```

To cancel an Order the document only requires an identifier and an action (Listing 4.10).

Listing 4.10 Generated cancel order "Order.XML" with proposed method

```
<order ID="0001" Action="Cancel">
</order>
```

Similar to Cancel Order, Response Order document (Listing 4.11) is very simple in that it only requires an element ResponseOrder.

Listing 4.11 Generated response order "Order.XML" with proposed method

```
<order ID="0001" Action="Response">
        <ResponseOrder>
                <AcceptedIndicator>true</AcceptedIndicator>
        </ResponseOrder>
</order>
```

Interaction result:

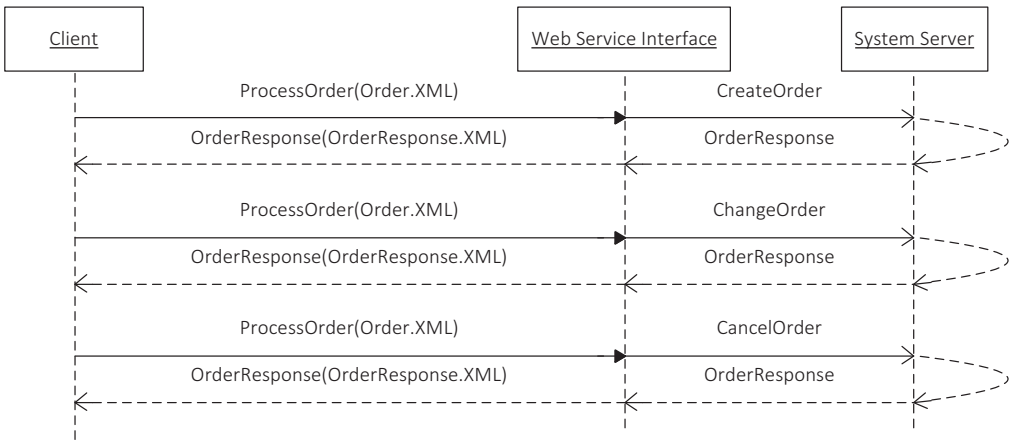


Figure 4.3 Proposed Service Interface

The proposed service interface (Figure 4.3) combines the three business functions into one Web Service operation and an OrderResponse operation. Both operations process the Order.XML. Therefore, according to the "Action" element, the system will determine which business functions are to be processed.

4.3 Advantages and Disadvantages

There are some advantages and disadvantages of using this approach, which are identified and discussed below (see examples and discussions in Chapter 6 (Section 6.2 and Section 6.3) for further details).

4.3.1 Advantages

Reusability

Reusability has been considered and addressed in the proposed methodology. First, the operation interface can be reused in many different contexts since it is designed based on a generic interface and adopting Core Component elements. Second, the proposed multi-grained operation is more reusable because it can now process the fine-grained (changeOrderLine – greater chance for reuse) and coarse-grained messages (changeOrder – reuse of changeOrderLine).

Minimalistic Design

The minimalist approach helps reduce the number of interfaces and elements to their simplest form. Minimising the amount of redundant data sent in the document achieves better throughput. Furthermore, the minimalist approach helps simplify the interaction result and minimises the impact due to the interface change that occurs.

Extensibility

Based on the Command pattern interface design, extensibility can be achieved by adding new actions in the business document. The operation interface is unchanged while adding or removing functionality. In contrast to the traditional approach, adding another function means creating a new operation in the new business document.

Flexibility

Flexibility is achieved by allowing multiple actions inside a business document. For instance, "update" and "cancel" actions can be put in the same changeOrder document to "update" and "cancel" any orderLine. This feature further enhances the efficiency of the sending document.

Maintainability

E-business application design with minimalist features will result in a smaller number of interfaces and fewer business documents, thus making them more manageable.

4.3.2 DISADVANTAGES

Longer Time to Implement

Developers need to understand the structure of the schema as well as the interface before they can adapt their program to the interface since it contains the element "Action" besides business content. Hence, this approach can take longer for the developer to create as an e-business application.

Re-develop Existing Application

Traditional e-business applications are exposed interfaces using business events. Existing architecture might not accept this proposed design message format. This scenario may require the developer to re-develop existing codes to process these documents.

Increase Complexity

This design approach reduces the number of interfaces by focusing on message design. The output messages will carry both business contents and processing instructions. Hence the complexity of the document will increase due to multiple actions within a document. Furthermore, there may be the possibility of sending incorrect actions for processing.

4.4 Discussion

The proposed service interface in Figure 4.3 follows the service-orientation design principles. These principles are discussed further in Section 6.3. The operation "ProcessOrder" is more reusable since it is handling multiple actions within a document compared to traditional operations: "OrderRequest", "ChangeOrder" and "CancelOrder" where each operation represents a business event. It was also designed with abstraction in mind since the operation interface is now more generic and can handle multiple actions within a document.

It can be reused for invoking multiple actions as well as other service calls in relation to an "order" business entity. Users can place a new order, change the order or cancel the order using the same "ProcessOrder" operation. The proposed interface can process fine-grained (i.e. single item) or coarse-grained (i.e. multiple items); this can maximise the reusability of the operation and make it ready for composition. Following the minimalist design approach can reduce the number of interfaces and elements to their simplest form. Reducing the number of interfaces not only helps simplify the interaction, but also keeps the interface stable if any change occurs due to the generic service operation. Thus the developer only needs to focus on the required schema design based on the provided business document and to adapt their program to the service interface.

The redefined business document also takes interoperability into account. The existing elements and the UBL structure can be used to redefine the business document with the proposed design method. The method relies on the UBL core components with the addition of an "action" attribute in order to simplify the service interaction. This re-engineered business document can be communicated with the existing service by employing a process-switching layer.

The minimalist service operations encapsulate process action and data from a single business entity (i.e. Order) to ensure the interface's stability. Exchanging of these business documents relies on the process-switching operation. This can imply control coupling between the service calls. Even though control coupling has a higher coupling

level than data coupling and stamp coupling since it contains a control parameter, it is still considered to be normal coupling and is therefore acceptable (Vinoski, 2005). This will also have a minimal impact on reusability since all service operations are stateless, and the messages' sequences are governed by the document identifier. The proposed operation can handle both "update" and "cancel" of its elements. It conforms to the atomic operation design, where the user can perform single or multiple actions depending on usage.

Business suitability is also considered in this methodology. With extensibility, new actions can be added to the business document while keeping the interface unchanged. Changes are only made through the selected element of a document. This is comparable to existing business document designs where updating documents requires a major revision since changed elements in a document might affect several other documents. Secondly, flexibility is achieved; different actions can be incorporated into a single message, greatly enhancing the efficiency of the message. Thirdly and finally, the issue of maintenance is addressed. There are many business documents as well as the interface in any domain-specific e-business applications; hence managing them becomes difficult. This approach helps to reduce the number of documents and interfaces, making the interface easier to manage.

Web Service interface is similar to the procedural method where the API is externalised for invocation. Therefore, existing principles applied to API design can be utilised as a reference for Web Service interface design. The most common principle is concerned with interdependence, such as coupling and cohesion. Coupling is concerned with the level of dependence between two modules, while cohesion is concerned with the relationship between internal modules. The principle suggests that reducing coupling while increasing cohesion can help software achieve higher reusability. In this research, we only consider normal coupling since this is acceptable for software design (Vinoski, 2005). Normal coupling consists of three types. The first form is data coupling where individual data elements are passed, which is similar to RPC. The second form is stamp coupling where the data structure is passed, therefore value, format and organisation must be matched, and this type is similar to Document Style. The third form is control coupling where one component passes parameters to control the activity of another

component, which is similar to the proposed method. Control coupling has a higher coupling level than data coupling and stamp coupling since it contains a control parameter. However, it is still considered to be acceptable coupling (Vinoski, 2005).

This thesis employs minimalism to help reduce the number of interfaces and to use fewer business documents to arrive at the Web Service interface design. There are some disadvantages in employing this method approach, in that developers need to understand the schemas as well as the interfaces before integrating them with their web application. It is also possible that developers need to re-develop existing applications to accept the messages. However, the ability to process multiple actions and reduce redundancy is more efficient for communicating between services, especially in domain-specific e-business applications where documents are large and complex. Furthermore, the composition of Web Services is now simplified since fewer interfaces are needed to start with. Another notable feature is extensibility where changing the business document now will not affect the composition pattern because the client is still using the same interface. By treating the document as an object, the developer can provide it with more functionality while minimising the interface by putting "Processing Instruction" into the business document. Lastly, the developer only maintains a smaller number of business documents for multiple business processes.

CHAPTER 5
IMPLEMENTATION

The previous chapter described and illustrated the proposed Web Service interface design methodology using a simple case study involving the UBL Ordering fulfilment specification document. In this chapter, we will illustrate the implementation of the proposed design methodology through a larger case study (Section 5.1), i.e. designing a Web Service interface for a purchase order business process from the UBL business document specification. We first describe and outline the specification requirements for the case study. In Section 5.2 we will outline the implementation details; the prototype application is designed using a three-tier architecture with the bottom tier being the database server using MySQL, while mid-tier is the business logic written in Java which handles all the purchase ordering processes. The top tier is the GUI (Graphic User Interface), which accepts and displays the messages.

5.1 Implementing Case Study

Ordering is a process that creates a contract business document between the Seller Party and Buyer Party, which is summarised in Table 5.1. Document types in these processes are Order, Order Response, Order Response Simple, Order Change and Order Cancellation.

Buyer's Business Process	Seller's Business Process	UBL Document
Place Order →	Receive Order	Order
Receive Response ←	Reject Order	Order Response Simple Order Response
Receive Response ←	Add Detail	Order Response
Receive Response ←	Accept Order	Order Response Simple Order Response
Change Order →	Change Order	Order Change
Cancel Order →	Cancel Order	Order Cancellation

Table 5.1 Summary of the Ordering business process

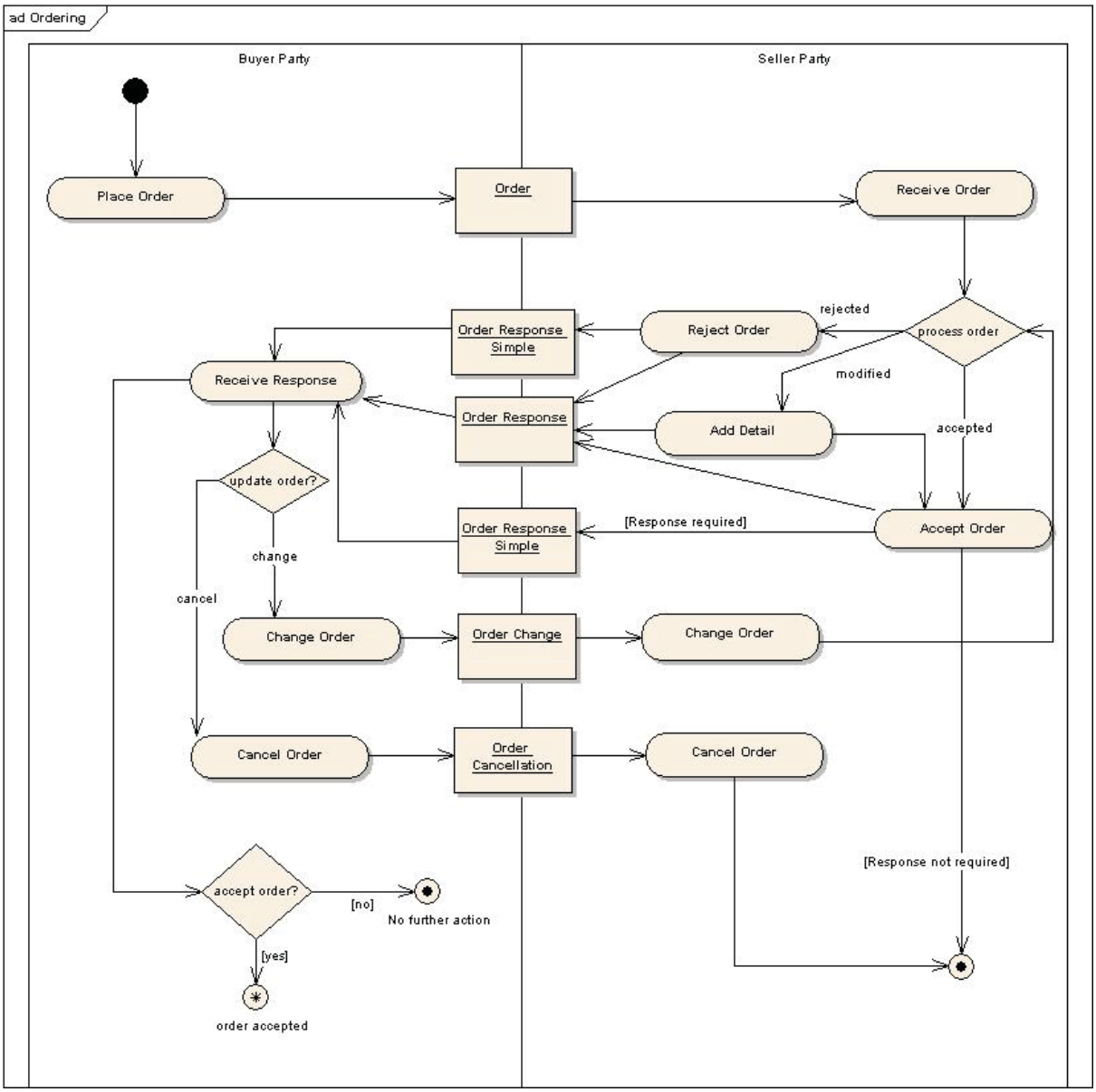


Figure 5.1 Fulfilment Case Study (Source: UBL)

5.1.1 INDIVIDUAL BUSINESS PROCESS AND REQUIREMENT

From the Fulfilment's UML diagram (Figure 5.1) above, we will describe several basic scenarios and requirements of order fulfilment.

Ordering Process:

The Order may specify allowance and charge instructions that identify the type of charge and who pays which charges. The Order may be placed on an account or credit/debit card account. The Order allows for an overall currency defining a default for all pricing and also a specific currency to be used for invoicing. Within an Order, additional currencies may be specified both for individual item pricing and for any allowances or charges. Refer to Appendix 14 for complete UBL PurchaseOrder xml message.

-Trade discount may be specified at the Order level. The Buyer may not know the trade discount, in which case it is not specified.

-The Order may provide multiple Order Lines.

-The Order may specify delivery terms, while the Order Line may provide instructions for delivery.

-The Buyer may indicate potential alternatives that are acceptable.

Listing 5.1 Place Order (Order.xsd)

```
<xs:element name="Order">
 <xs:complexType>
  <xs:sequence>
        <xs:element name="ID" type="xs:string" />
        <xs:element name="IssueDate" type="xs:date" />
        <xs:element name="BuyerCustomerParty" type="CustomerParty" />
        <xs:element name="SellerSupplierParty" type="SupplierParty" />
        <xs:element name="OrderDetail" type="OrderLine" />
  </xs:sequence>
 </xs:complexType>
```

Listing 5.2 Order example

```
<Order >
<cbc:ID>65830</cbc:ID>
<cbc:IssueDate>2011-11-01</cbc:IssueDate>
+ <cac:BuyerCustomerParty />
+ <cac:SellerSupplierParty />
+ <cac:OrderLine />
</Order>
```

The Order Response Simple is the means by which the Seller confirms receipt of the Order from the Buyer, indicating either commitment to fulfil the Order without changing or rejecting the Order. Refer to Appendix 15 for the complete UBL OrderReponse xml message

Listing 5.3 Reject Order (Order Response Simple.xsd) and Accept Order (Order Response Simple.xsd)

```
<xs:element name="OrderResponseSimple">
  <xs:complexType>
   <xs:sequence>
        <xs:element name="ID" type="xs:string" />
        <xs:element name="OrderReference" type="xs:string" />
        <xs:element name="IssueDate" type="xs:date" />
        <xs:element name="AcceptedIndicator" type="xs:string" />
        <xs:element name="BuyerCustomerParty" type="CustomerParty" />
        <xs:element name="SellerSupplierParty" type="SupplierParty" />
   </xs:sequence>
  </xs:complexType>
```

Listing 5.4 Reject OrderResponseSimple example

```
<OrderResponseSimple >
<cbc:ID>65830</cbc:ID>
<cbc:IssueDate>2011-04-02</cbc:IssueDate>
<cbc:AcceptedIndicator>false</cbc:AcceptedIndicator>
<cbc:RejectionNote>Out of Stock</cbc:RejectionNote>
+ <cac:OrderReference />
+ <cac:Signature />
+ <cac:SellerSupplierParty />
+ <cac:BuyerCustomerParty />
+ <cac:AccountingSupplierParty />
+ <cac:AccountingCustomerParty />
</OrderResponseSimple>
```

The Order Response proposes to replace the original Order. It reflects the entire new state of an order transaction. It is also the means by which the Seller confirms or supplies Order-related details to the Buyer who was not available to, or specified by the Buyer at the time of ordering.

Listing 5.5 Reject Order (Order Response.xsd) and Add Detail (Order Response.xsd) and Accept Order (Order Response.xsd) - with modified detail

```
<xs:element name="OrderResponse">
  <xs:complexType>
   <xs:sequence>
        <xs:element name="ID" type="xs:string" />
        <xs:element name="OrderReference" type="xs:string" />
        <xs:element name="IssueDate" type="xs:date" />
        <xs:element name="BuyerCustomerParty" type="CustomerParty" />
        <xs:element name="SellerSupplierParty" type="SupplierParty" />
        <xs:element name="OrderDetail" type="OrderLine" />
   </xs:sequence>
  </xs:complexType>
 </xs:element>
```

Listing 5.6 Reject OrderResponse example

```
<OrderResponse >
<cbc:ID>65830</cbc:ID>
<cbc:IssueDate>2011-04-01</cbc:IssueDate>
+ <cac:OrderReference />
+ <cac:SellerSupplierParty />
+ <cac:BuyerCustomerParty />
+ <cac:OrderLine />
</OrderResponse>
```

The Buyer may change an established Order in two ways, subject to the legal contract or trading partner agreement: first, by sending an Order Change; or second, by sending an Order Cancellation followed by a new, complete replacement Order.

An Order Change reflects the entire current state of an order transaction. Buyers may initiate a change to a previously accepted order for various reasons, such as changing ordered items, quantity, delivery date and ship-to address. Suppliers may accept or reject the Order Change using either Order Response or Order Response Simple. Refer to Appendix 16 for complete UBL OrderChange xml message.

Listing 5.7 Change Order (OrderChange.xsd)

```
<xs:element name="OrderChange">
  <xs:complexType>
   <xs:sequence>
         <xs:element name="OrderReference" type="xs:string" />
        <xs:element name="IssueDate" type="xs:date" />
        <xs:element name="SequenceNumberID" type="xs:string" />
        <xs:element name="BuyerCustomerParty" type="CustomerParty" />
        <xs:element name="SellerCustomerParty" type="SupplierParty" />
        <xs:element name="OrderDetail" type="OrderLine" />
   </xs:sequence>
  </xs:complexType>
 </xs:element>
```

Listing 5.8 OrderChange example

```
<OrderChange>
<cbc:ID>65830</cbc:ID>
<cbc:IssueDate>2011-04-10</cbc:IssueDate>
<cbc:SequenceNumberID>01</cbc:SequenceNumberID>
+ <cac:OrderReference />
+ <cac:BuyerCustomerParty />
+ <cac:SellerSupplierParty />
+ <cac:OrderLine />
</OrderChange>
```

At any point in the process a Buyer may cancel an established order transaction using the Order Cancellation document. Refer to Appendix 17 for the complete UBL OrderCancellation xml message

Listing 5.9 Cancel Order (OrderCancellation.xsd)

```
<xs:element name="OrderCancel">
  <xs:complexType>
   <xs:sequence>
        <xs:element name="ID" type="xs:string" />
        <xs:element name="OrderReference" type="xs:string" />
        <xs:element name="IssueDate" type="xs:date" />
        <xs:element name=" CancellationNote" type="xs:string" />
        <xs:element name="BuyerCustomerParty" type="CustomerParty" />
        <xs:element name="SellerSupplierParty" type="SupplierParty" />
   </xs:sequence>
  </xs:complexType>
 </xs:element>
```

Listing 5.10 OrderCancellation example

```
<OrderCancel>
<cbc:ID>65830</cbc:ID>
<cbc:IssueDate>2011-04-01</cbc:IssueDate>
<cbc:CancellationNote>Out of Stock</cbc:CancellationNote>
+ <cac:OrderReference />
+ <cac:BuyerCustomerParty />
+ <cac:SellerSupplierParty />
</OrderCancel>
```

5.1.2 PROPOSED ORDERING PROCESS

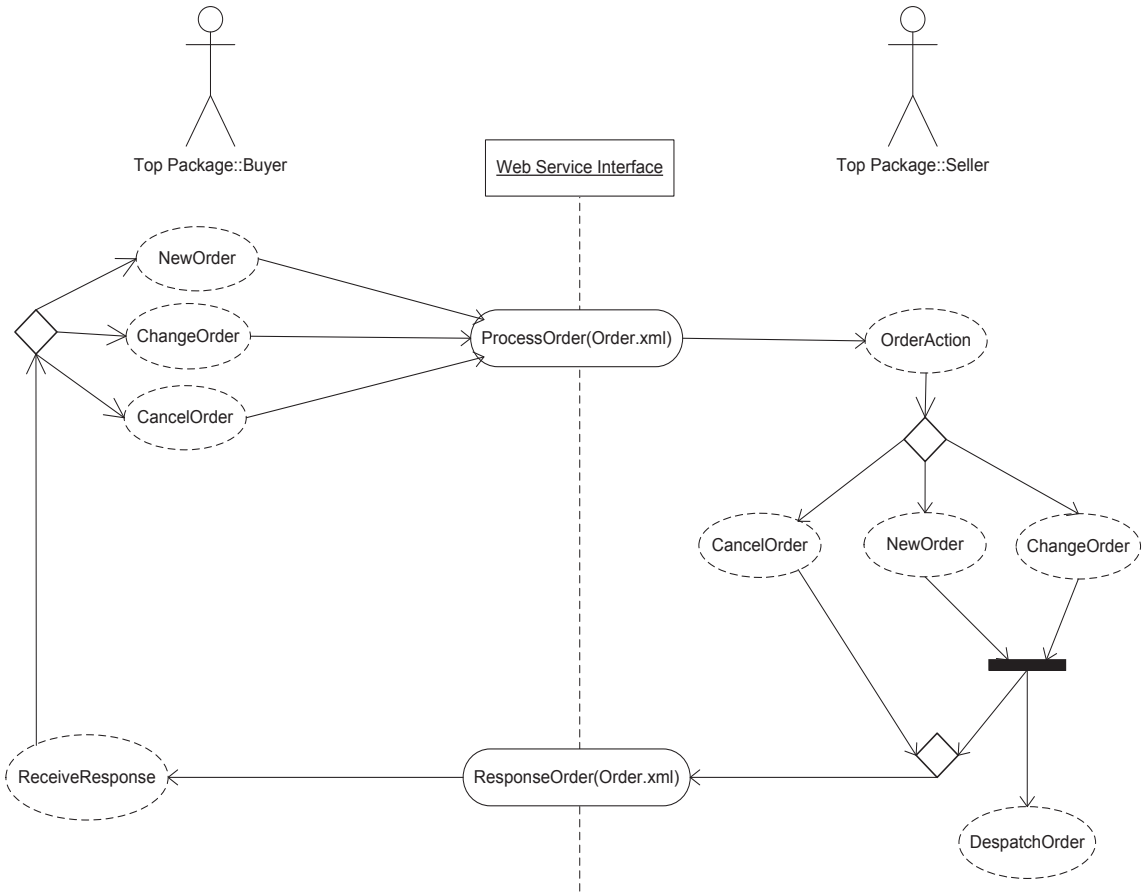


Figure 5.2 Proposed Ordering use case diagram

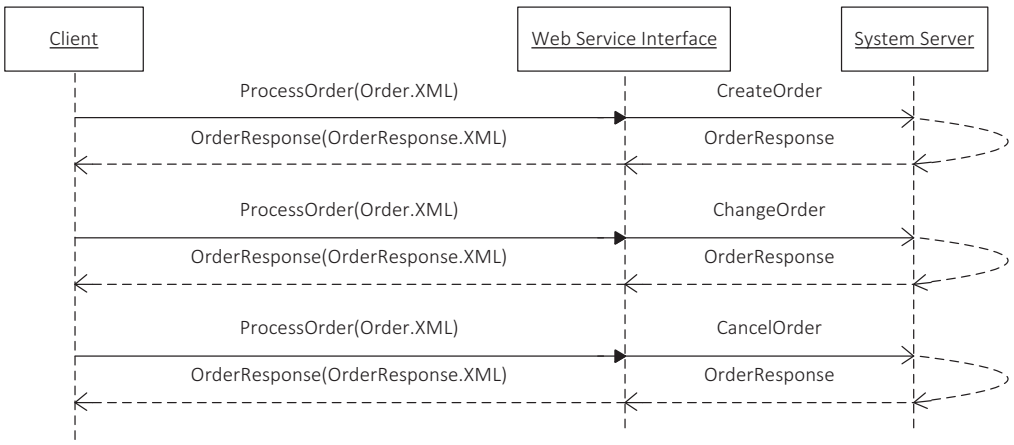


Figure 5.3 Sequence diagram of proposed use case

5.2 Implementation Detail

5.2.1 System Architecture

The application prototype is developed based on three-tier architecture (Figure 5.4) for scalability. The bottom tier is the database server, which stores all the required data for the order processing (refer to Section 5.2.2). The middle tier is the application server, which processes the request and produces the response Web Service message. This tier contained all the business logic on how to process an order, whether to accept or reject an order and is also responsible for error handling (refer to Section 5.2.3). The top tier is the graphic user interface; this tier is responsible for the interaction between the user and application, accepting the user input SOAP message and displaying the server response message (refer to Section 5.2.4).

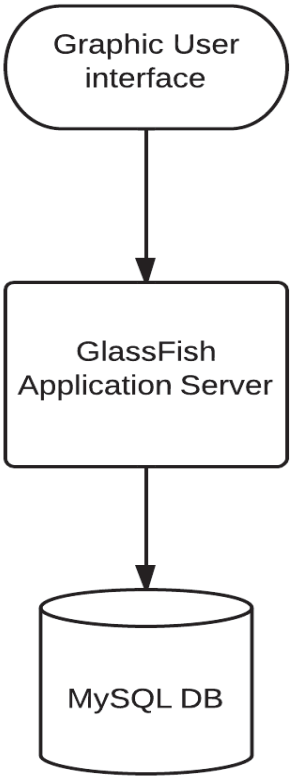


Figure 5.4 Prototype System Architecture

5.2.2 DATABASE SYSTEM

MySQL Server 5.0 is utilised as our main database server. Database tables and elements are created based on the process Order Fulfilment from UBL. The script used to create the tables is documented in Appendix 1. The following entity relationship diagram (Figure 5.5) outlines the entities' details and their relationship.

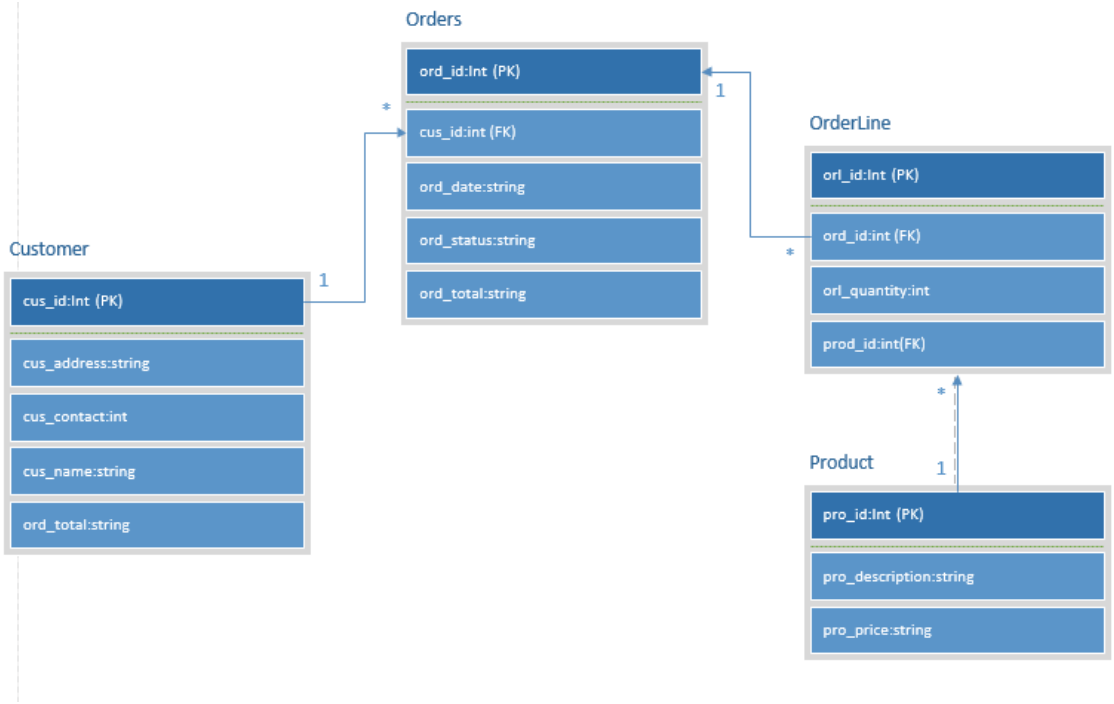


Figure 5.5 Database Table Diagram

5.2.3 APPLICATION SERVER SYSTEM

We use Netbean 6.8 to develop the prototype application and Glassfish server to run our Java Enterprise Edition and Web Service application. The following is the class diagram of the case study (Figure 5.6), while the data structure and class diagram will be the same for UBL and the proposed process. Since we are only working on the remote interface, our system has been design based on transactional interaction; each remote call will be a transaction. Any unsuccessful calls will be terminated and rolled back. See Appendix 2 for more information about the individual method call.

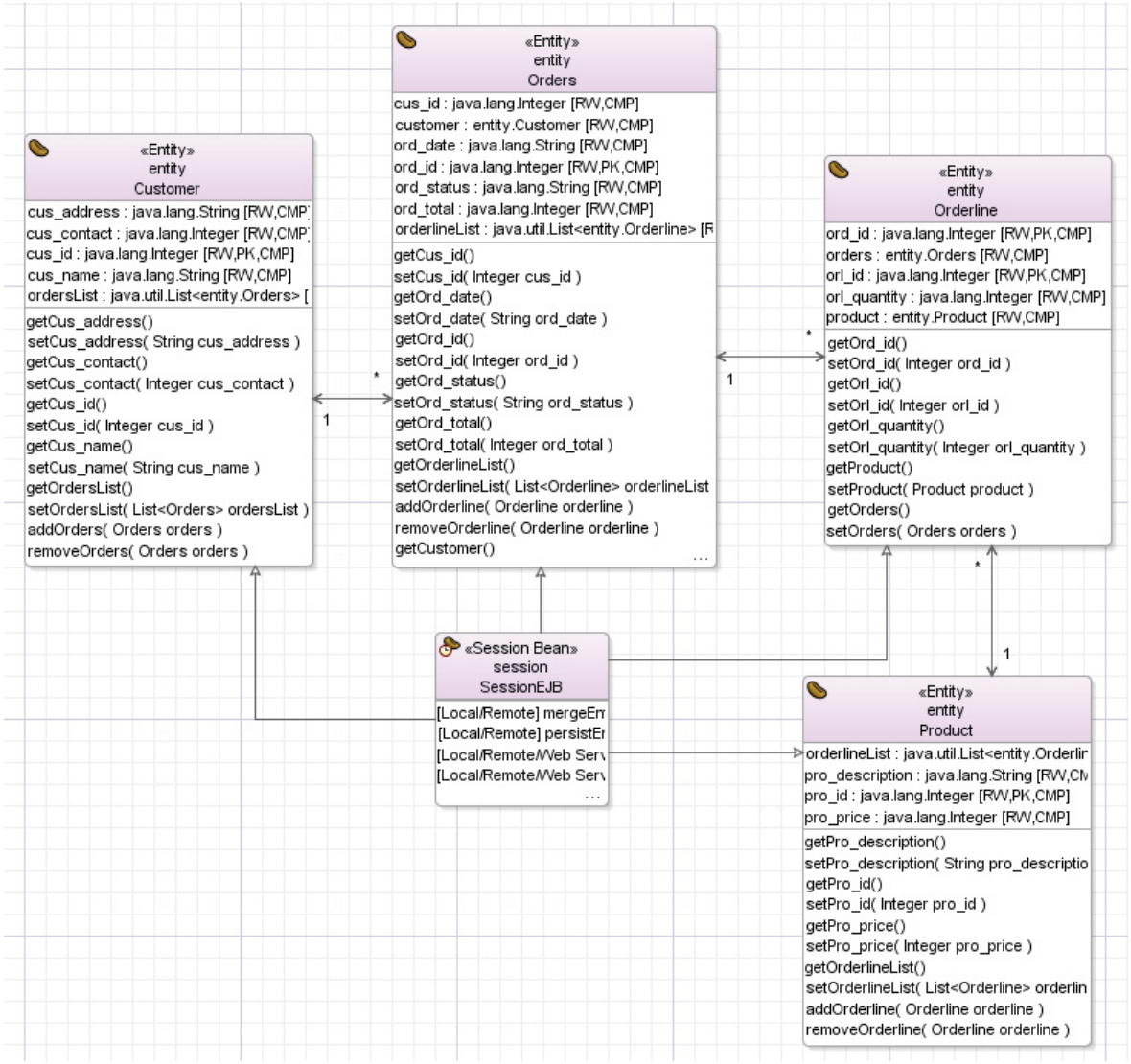


Figure 5.6 Overall Class Diagram of the Case Study

5.2.4 APPLICATION PROTOTYPE

Figures 5.7 - 5.10 represent the demo prototype of the Web Service invoking an interface. The left panel represents the service request from the user in the XML format (i.e. Order Business Document). The right hand side presents the service response from the application server. When the user clicks "Send", the prototype application will accept the request from the user and invoke the Web Service, and then display the response on the right hand side.

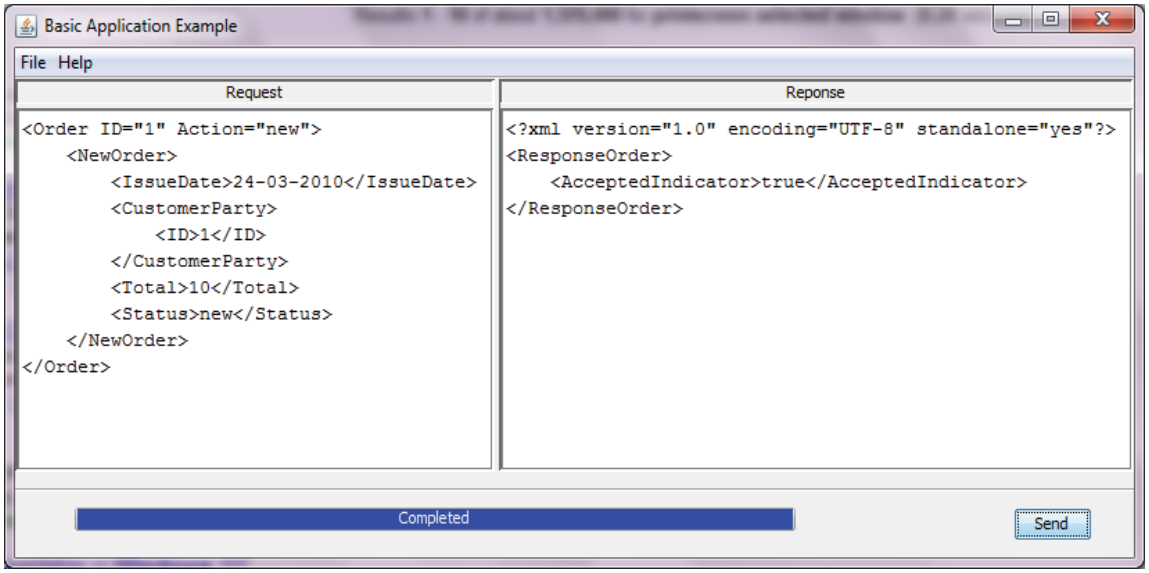


Figure 5.7 Interface of the test new order

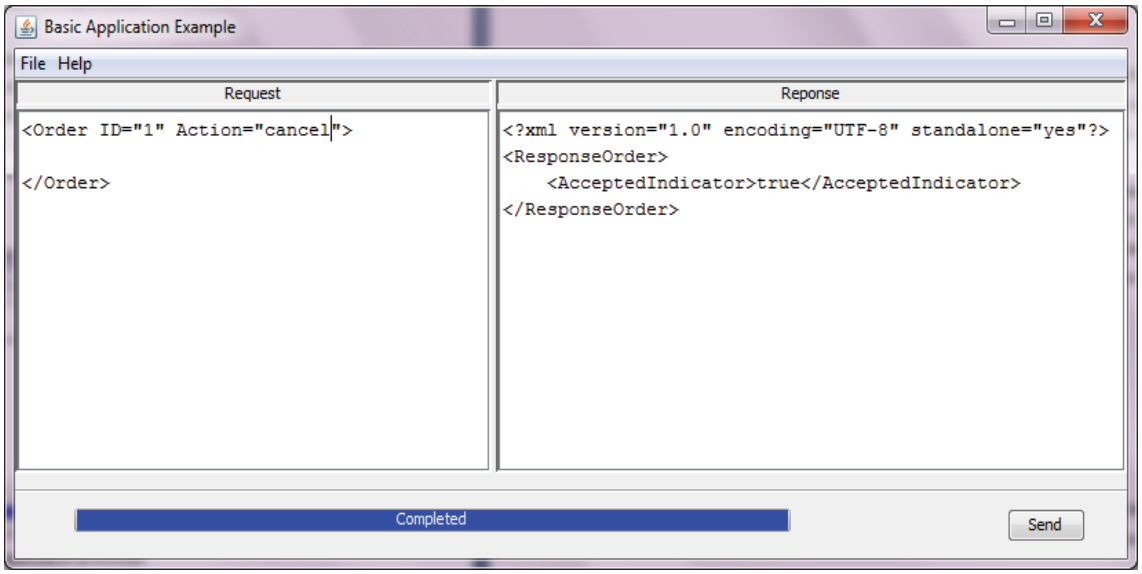


Figure 5.8 Interface of the test cancel order

Figure 5.7 shows a new order being created and accepted by the system and Figure 5.8 shows an order being cancelled.

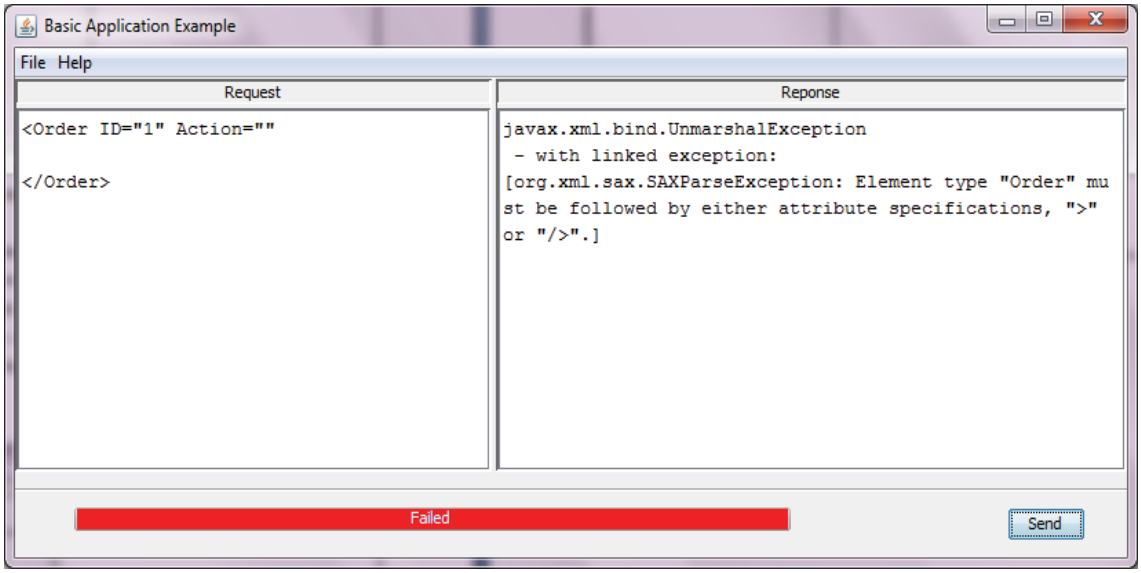


Figure 5.9 Exception error input from the user

Figure 5.9 illustrates an exception thrown by the Web Service Response

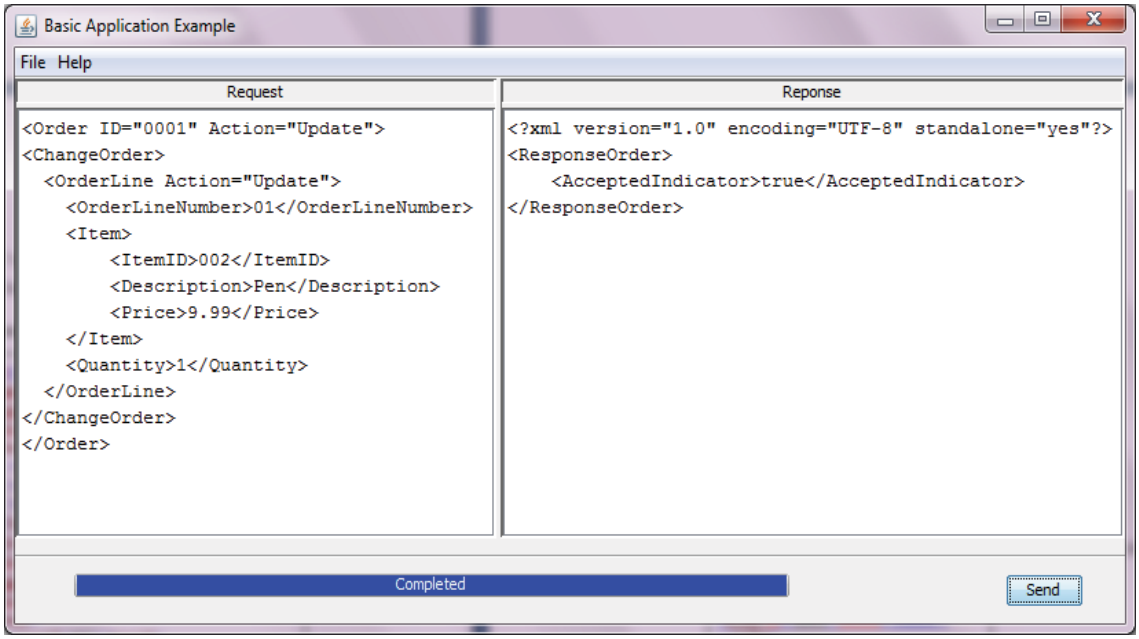


Figure 5.10 Interface of the test update order

Figure 5.10 depicts a successful update of an order.

CHAPTER 6
EVALUATION

In Chapter 5 we described and outlined the implementation of the proposed Web Service interface design methodology. We illustrated a simple case study based on the UBL Ordering fulfilment specification document. In this chapter, we evaluate the proposed methodology using the results of the implementation compared to the UBL examples listed in Chapter 5. The evaluation criteria are based on the Service-Oriented design principles written by Thomas Erl and another relevant study by Legner (2007) as shown in Section 6.1. Specifically, the focus is on the following principles - reusability, abstraction, and business suitability. We first describe and outline the service-oriented design principles for the evaluation. In Section 6.2, a quantitative approach evaluation is presented to compare the results from the minimalist design approach with the standard UBL approach. In Section 6.3, a qualitative approach evaluation will be conducted to verify that the proposed document design conforms to the principles of the Service-Oriented Design (as stated in Section 6.1).

6.1 Service-Oriented Design Principles

The focus of the proposed methodology is on the design of service interfaces based on minimalism while adopting the design principles derived from studies on Web Services. In order to create a well-designed interface, a set of design principles must be taken into account. According to Legner (2007) and Erl (2008), these service design principles are described in more detail below.

Standardised Service Contract: The service interface defines its purpose and capability; therefore, a standard and compliant service contract is important to the external party. A standardised service contract includes identifying the service's name, endpoint description and its elements in the document schema (these rely on the context neutral Core Component Specification). Web Services depend on open and industry standards for interoperability. Thus, organisations must design their services so that they conform to industry technical and business standards such as the promotion of the domain-specific business document. This principle will support the interoperability of services, resulting in a better client-driven service.

Loose Coupling Principle: This principle refers to the relationship or dependency between two services. It is recommended that the coupling level should remain low in order to promote the independence and evolution of a service's logic and implementation. The result should be a more reusable and easy-to-integrate interface. It follows the traditional software component design approach, such as maximising functional cohesion and minimising coupling in order to reduce the services' dependence on each other.

Abstraction: Web Service interface should be designed with abstraction from service implementation detail. Service interface specification should be uniform and comprehensively defined, so that it produces a stable and managed service contract. The abstraction principle indicates that the details of software artefacts, which are not required, should be hidden or removed. Therefore, all the information necessary to invoke the service is contained in the service contract and provided by the schema. The abstraction principle enables replaceability and enhances scalability.

Reusability: The reusability principle states that the service interface should serve as an enterprise resource, which is agnostic to the functional context. The logic of a service should be highly generic in order to reuse and apply it in various contexts. The service logic can be repeated and reused in multiple scenarios.

Autonomy: The autonomy principle states that the services resulting output can only be determined by the input of the same service contract. The service contract should produce the output regardless of any external influence such as system performance, system platform and integration application. Service autonomy improves the reliability of the service contract.

Statelessness Principle: The statelessness principle states that the services should avoid or minimise the management of state information to minimise unnecessary resource consumption. Stateful service can consume more resources, increase complexity and reduce reusability due to the requirements of state information. The statelessness principle can improve service scalability.

Discoverability Principle: This principle states that we should annotate services with metadata in service descriptions so that service discovery can be more effectively.

Composability Principle: The composability principle identifies services as effective composition participants, regardless of their size and/or complexity. Each service should be treated and designed as a composable entity. A complex business process can compose using different services, avoiding the need to redevelop an existing service, and thus reduce costs and time.

Business suitability: Granularity that balances the number of operations a Web Service should be well chosen. The developer should determine a more complete requirement and consider the potential evolution of a service, which can evolve to respond to the new requirements in a controlled manner.

In particular, the proposed methodology for the web service design concentrates on the key principles of reusability, business suitability, and abstraction to differentiate itself from other methodologies, such as from UBL. These principles will be evaluated in-depth in section 6.3, along with other benefits such as extensibility, flexibility, maintainability. The other principles mentioned such as Standardised Service Contract, Loose Coupling Principle, Autonomy, Statelessness, and Discoverability are common

industry practice, thus do not require further evaluation and they are out of scope of the thesis objectives.

6.2 Quantitative Evaluation

The quantitative evaluation section will carry out some experiments to analyse, evaluate and compare the results from the proposed Web Services design against the UBL approach. Multiple purchase order scenarios for different number of items will be invoked using both Web Services approach. We then compare and analyse the results based on the message sizes and the response time.

We will set up a standard machine with Intel Core Duo 2.4 GHz processor, 4 GB of RAM and running Windows 7. The application is hosted on the Sun Glass Fish Application server, MySQL 5 Database server and Netbean 6.8 as the development and testing environment. We have implemented both service invoking methods on the same machine with the same application server. Results will simply compare the two approaches for evaluation purposes only. The outcomes will not indicate the real life performance of the application, as this depends on server hardware, Internet bandwidth and location of invocation. In order to invoke the Web Service, we have created a separate Client project (Figure 6.1) to measure the performance. The client is using the proxy generated from the published Web Service WSDL for invocation.

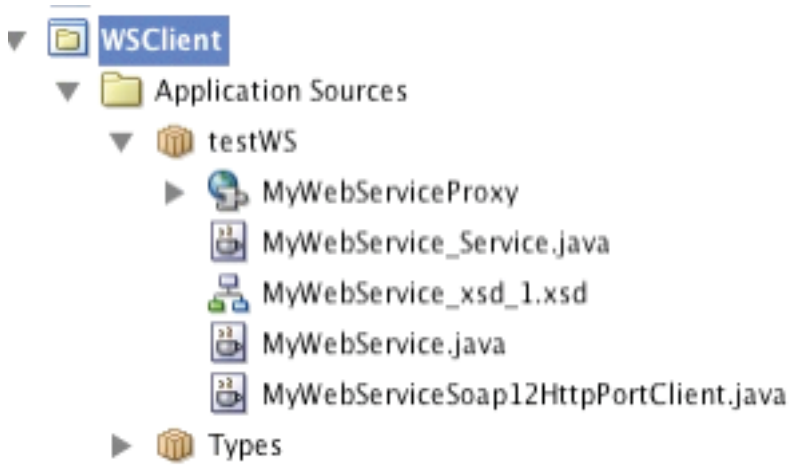


Figure 6.1 Client project for evaluation

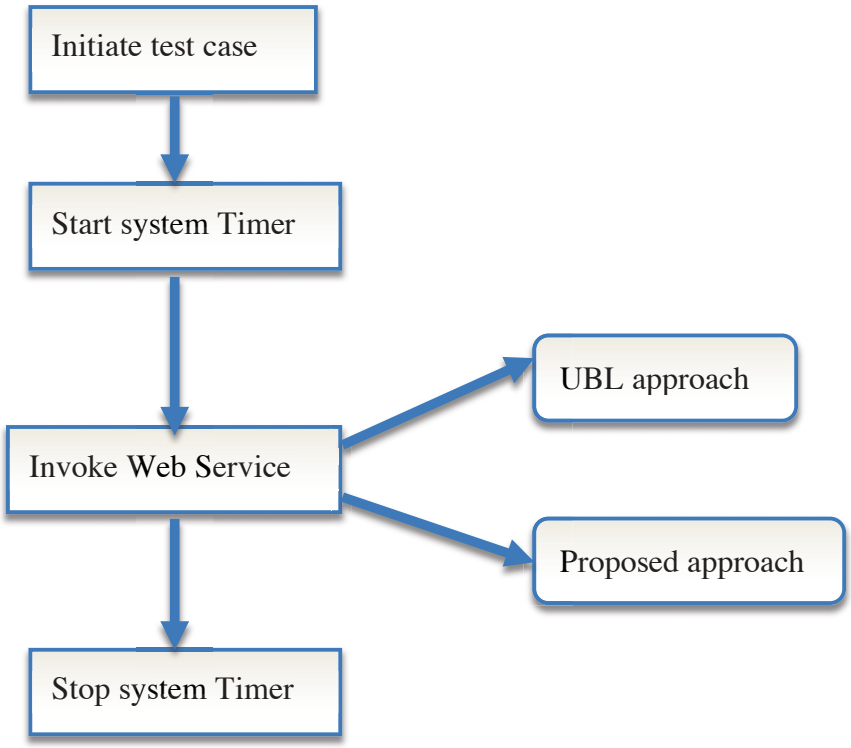


Figure 6.2 Client testing procedure

We will set up several test cases for PurchaseOrder, ChangeOrder and CancelOrder. A purchase order (see Listing 5.2 for the purchase order message) may contain 1 item, 3 items or 10 items. We will measure the elapsed time and the SOAP message size for evaluation.

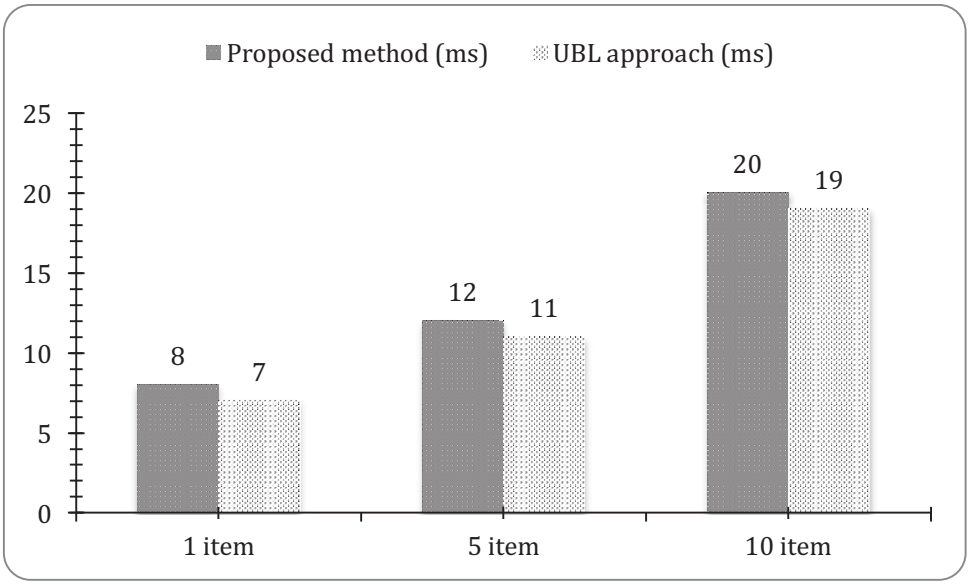


Figure 6.3 Comparison of response time (ms) between proposed method vs. UBl approach for New PurchaseOrder

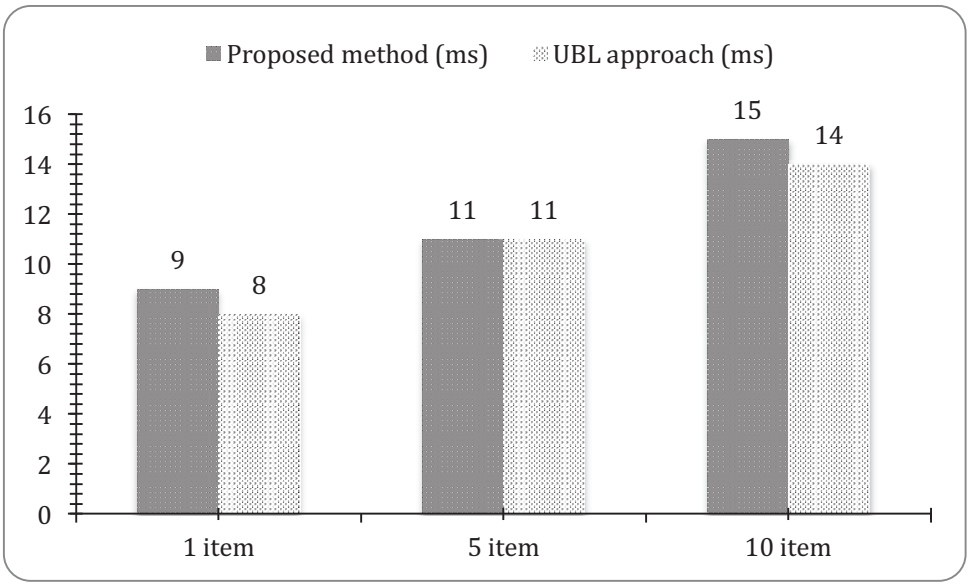


Figure 6.4 Comparison of response time (ms) between proposed method vs. UBl approach for Cancel PurchaseOrder

In Figures 6.3 and 6.4 the results indicate that there is not much difference in response time between these two approaches for a new and cancel order. This is because they both require the same information for processing. The proposed method does require extra time (1 ms) in the action switch-processing layer. Overall the difference between the two approaches for new and cancel order is negligible (refer to Listing 5.10 for the orderCancellation message).

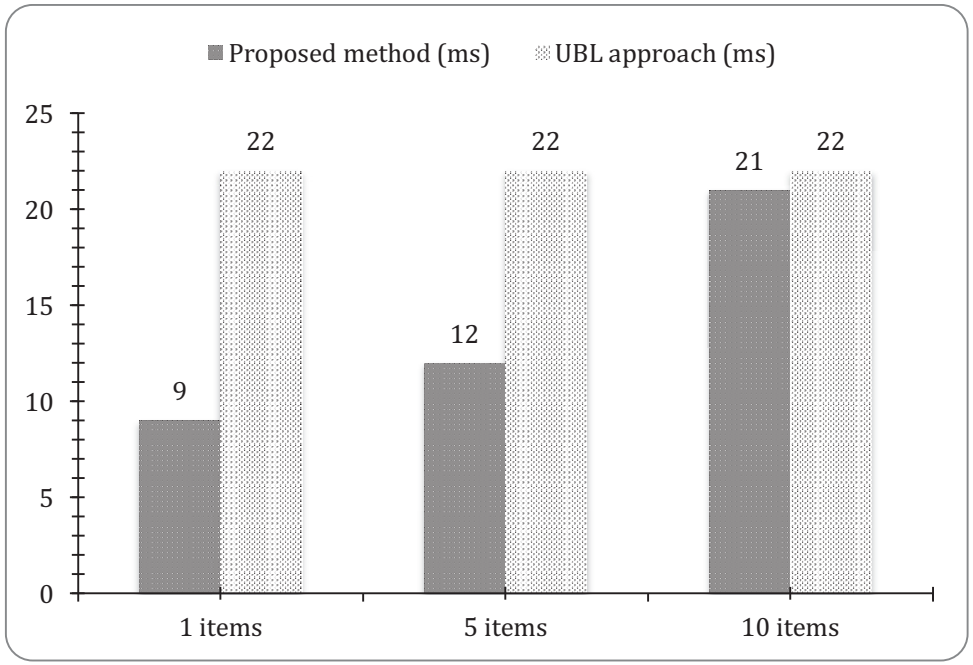


Figure 6.5 Comparison of response time (ms) between proposed method vs. UBl approach for Change PurchaseOrder with 10 items

For changing the PurchaseOrder refer to Listing 5.8 for the orderChange message. In Figures 6.5 and 6.6 we use a PurchaseOrder with 10 items, then subsequently changing this to 1 item, 5 items or 10 items. The results indicate there is not any difference in response time or SOAP size using the UBL approach for changing 1 item or 10 items in a PurchaseOrder. This is because changing a UBL PurchaseOrder requires a resubmitted new version of the PurchaseOrder. Eventually the change in PurchaseOrder will be processed again. However, for the proposed method we are only required to submit the change in the PurchaseOrder, which greatly increases the response time and reduces the SOAP message size of the Web Services for processing the same information. In this example the response time and SOAP size is reduced by up to 70%.

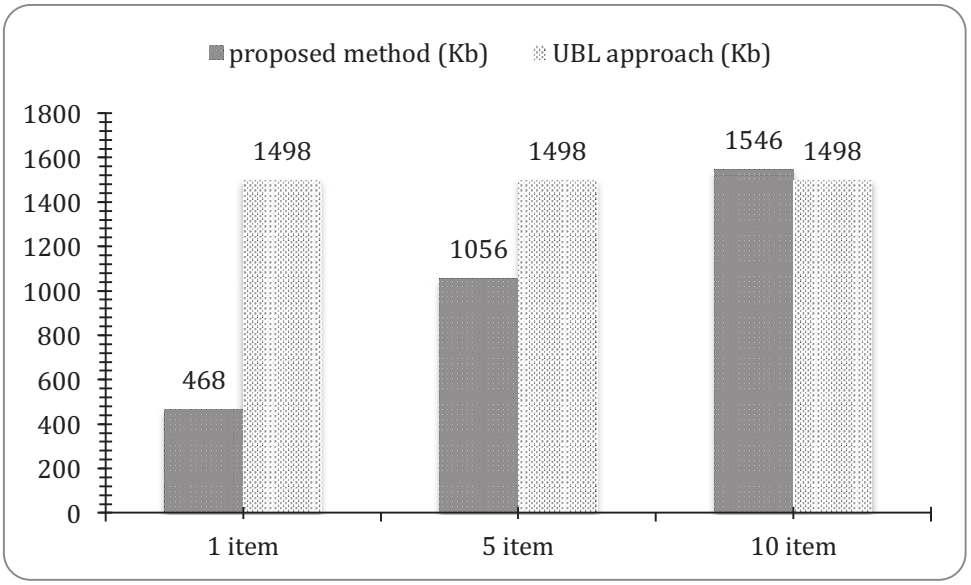


Figure 6.6 Comparison of SOAP message size (Kb) between proposed method vs. UBl approach for Change PurchaseOrder with 10 items

6.3 Qualitative Evaluation

The qualitative evaluation presented below will assess the proposed document design to verify if the proposed design conforms to the principles of the Service-Oriented Design including: reusability principle, business suitability, abstraction principle, extensibility, flexibility and maintainability. Each principle will be evaluated and compared using both the proposed document designs and the reference UBL documents.

6.3.1 Reusability Principle

This principle helps avoid duplication of the code by making the business process similar. It also helps achieve business agility by replacing a new business process from existing services to meet changing marketplace needs (Dan, 2008). By reusing an existing service, developers are able to reduce development time as well as to reduce costs, and improve the reliability of the service since reused business processes are well tested. Feuerlicht (2006) defines service reuse as the ability to participate in multiple service assemblies. Hence any Web Service interface must be designed so that it can compose or be composed by other services. The author also recommends using fine-grained and a unified message over coarse-grained message for greater reusability. Erl (2007) defines reusability service as something that should contain and express agnostic

logic, which enables service logic to be repeatedly leveraged over time. The logic encapsulated by the service is associated with a context that is sufficiently agnostic to any one-usage scenario. The author proposes that the logic encapsulated by the service is a sufficiently generic and extensible contract. In fact, the service contract is flexible enough to process a range of input and output messages. The service logic can be accessed concurrently to facilitate simultaneous access by multiple clients. Glusko (2005) proposed building the business document based on Core Component Type in order to increase reusability. Based on Document engineering, the business document must be generic and context-free so that it can be implemented in multiple scenarios.

Test Case	**Case Study Description**
Standard, Generic and Context-Free	Core component Type Based on UBL, can be used in more than one scenario and is being adopted by the majority of the industry partners.
Composable	Compose PurchaseOrder with DeliveryProcess.
Reusability	Fine-grained OrderLine can be reused for other purposes.

Table 6.1 Evaluation test case for reusability principle

The proposed operation "ProcessOrder" (Table 6.2) is designed to be reusable due to a number of factors. First, the service operation is designed to be generic, abstract and context-free. The service can be reused in any other business scenarios other than "purchasing" such as in a ticketing order service. Second, all elements in the proposed operation are based on the standard Core Component Type which are readily available and can be processed by the majority of partners. Third, the proposed service is more reusable by adopting the multi-grained approach, i.e. the message can be either fine-grained or coarse-grained. Since it is handling multiple actions within a document compared to traditional operations: "OrderRequest", "ChangeOrder" and "CancelOrder" where each operation represents a business event. This abstract interface will be reused for invoking multiple actions as well as other service calls in relation to an "Order" business document. Users can place a new order, a change order or cancel the order using the same "ProcessOrder" operation. Now a developer only needs to focus on the

required message/schema design based on the provided business document and adapt their program to the service interface.

Business Process	Traditional UBL operation	Proposed Operation
Send New Order	PurchaseOrder(OrderRequest.xml)	ProcessOrder(Order.xml)
Send Update Order	ChangeOrder(OrderChange.xml)	
Send Cancel Order	CancelOrder(OrderCancel.xml)	
Send Response Order	ResponseOrder(OrderResponse.xml)	ResponseOrder(Order.xml)

Table 6.2 Proposed minimalist operation

Test Case: There is only one interface ProcessOrder(order.xml) (Figure 6.7) between the systems, so a client can specify the "action" as new, cancel or update an order. The client can also specify which elements will be included as shown below. Element "Orderline" can be added/removed dynamically upon use.

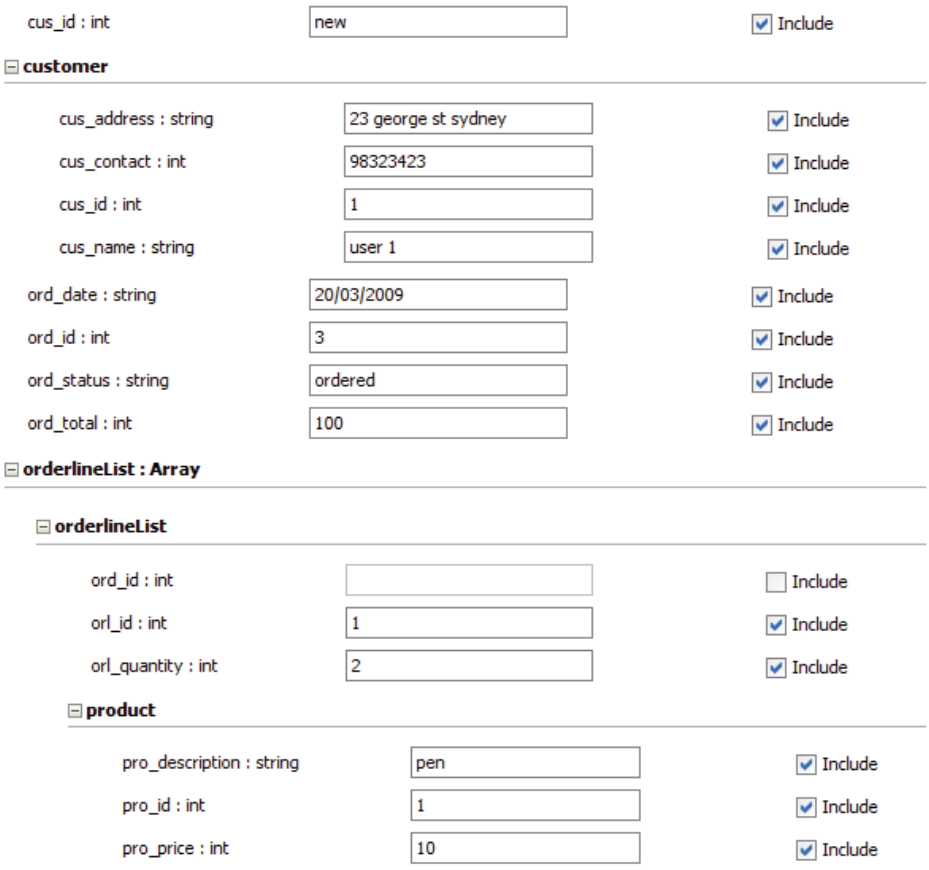


Figure 6.7 Web Service interface with processing of new order

6.3.2 BUSINESS SUITABILITY

This balances the number of operations a service should have. Steps involved are defining the input/output parameters, and grouping them to form an operation. Fine-grained service typically corresponds to elementary business function and implements highly reusable business logic (Bieberstein, 2005). Coarse-grained services typically correspond to individual business functions and carry their associated documents. However, it should be noted that choosing the correct level of granularity is based on an individual service requirement rather than a certain level of granularity to facilitate reuse. For instance, in a purchase order scenario, a fine-grained approach is to expose the operation down to individual create "order line". A coarse-grained approach is to expose the operation at a higher-level such as a request "PurchaseOrder". The fine-grained approach can vastly facilitate reusability. However, this is impractical and inappropriate due to multiple invocations required to complete a purchase order. On the other hand, to update an "order line", the fine-grained approach is more efficient than the coarse-grained one due to updating the online line item. Therefore, we allow multi-grained operation service. By following the minimalist approach, we only send the necessary data in the operation. Granularity will vary when different operations are employed.

Test Case	**Case Study Description**
Test Multi-Grained Service, Different messages utilised different levels of granularity.	-ProcessOrder(newOrder.xml) → Coarse Grained -ProcessOrder(cancelOrder.xml) →Fine Grained -ProcessOrder(ChangeOrder.xml) → Multi Grained - Send single or more items.

Table 6.3 Evaluation test case for well chosen granularity

Test Case: We can specify changing or removing any specific "orderline" (Figure 6.8) with the corresponding action in an order.

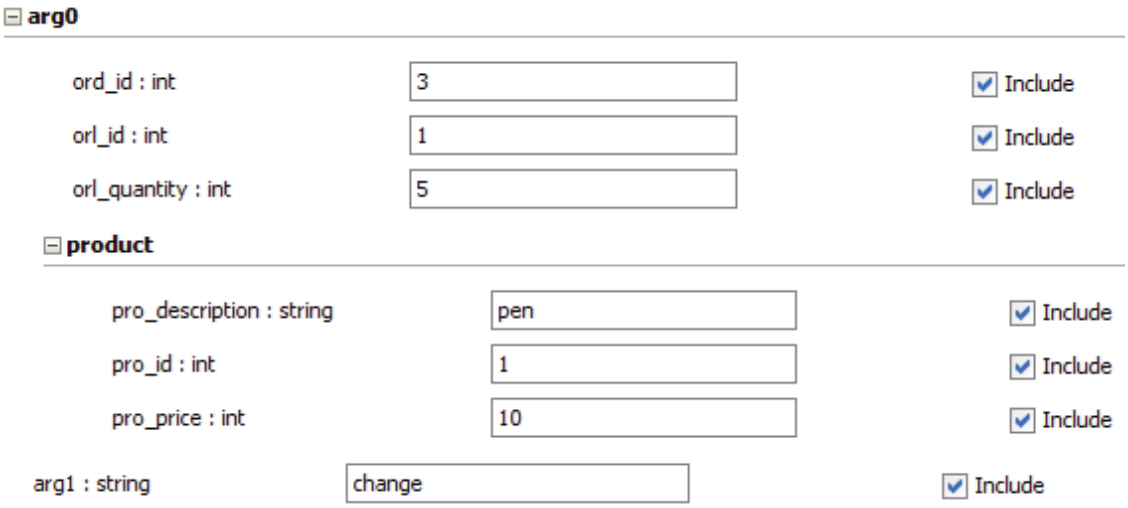


Figure 6.8 Web Service interface with processing change order

6.3.3 ABSTRACTION PRINCIPLE

The abstraction principle indicates that the details of software artefacts, which are not required, should be hidden or removed (Erl, 2007). Therefore, all the information necessary to invoke the service is contained in the service contract and provided by the schema. The abstraction principle enables replaceability and enhances scalability. Since the WSDL document provides a comprehensive and uniformly defined service specification, it also designs with abstraction since the operation interface can handle multiple actions within a document. It will be reused for invoking multiple actions as well as other service calls in relation to an "order" business document. The interface creates more stability due to the generic service operation, so the developer only needs to focus on the required schema design based on the provided business document and adapt their program to the service interface.

Following the minimalist approach to reduce the number of interfaces and elements to their simplest form (Figure 6.10), reducing the number of interfaces not only helps simplify the interaction outcome, but also keeps the interface stable if any change occurs. This can further reduce the document's size since only it is only sending the required data. Interaction becomes simpler because there are only two operations - back and forward.

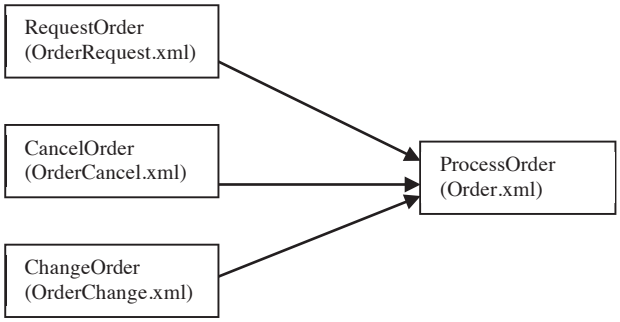
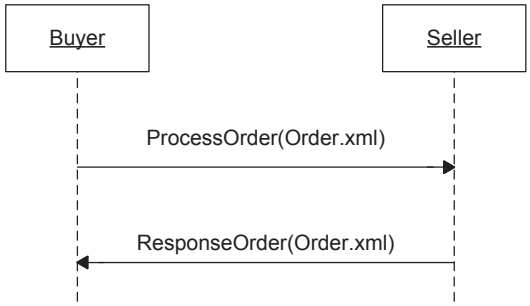


Figure 6.9 Minimalist interface approaches

Test Case	**Case Study Description**
Verify that the proposed interface is abstraction	-ProcessOrder(order.xml) can process new, cancel, update order

Table 6.4 Evaluation test case for abstraction principle



Test Case: With a minimalist interface we are only required to specify the XML document to be processed in the interface. Any system can produce and consume the XML document.

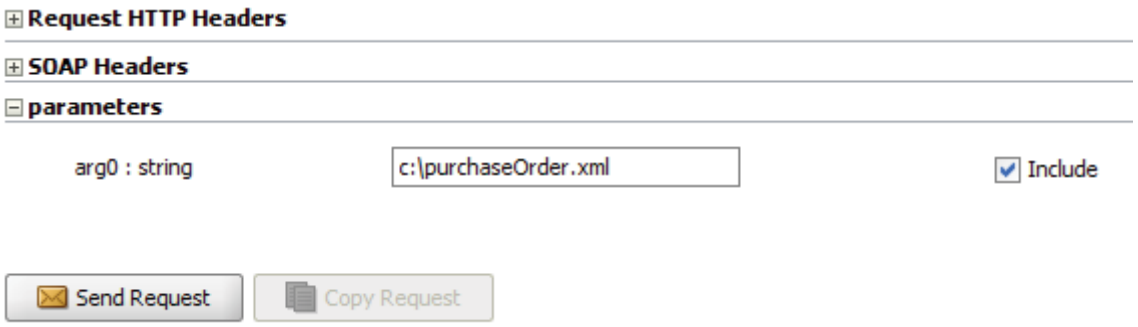


Figure 6.10 Web Service interface with processing order.xml

6.3.4 EXTENSIBILITY

Extensibility can be achieved by adding new functionality to the existing interface. With the proposed interface, new functions or logics (i.e. new actions) can be added to the business document while keeping the interface unchanged. Changes are only made through the selected elements of a document. Hence, this approach is fully compatible with new and existing interface.

In contrast to the current design of the business document, any changes to the document will require a new version as any elements changes may affect several others. Adding new functionality means introducing new operations to the interface and will require a new set of business document. This is an important feature because e-business is constantly evolving. Below is a demonstration of adding a new action to the Order document while using the existing interface (Table 6.6).

Test Case	Case Study Description
Add/delete new action to the existing interface without affecting existing operation	-Modify ProcessOrder(order.xml) -Add Retrieve Action -Add Retrieve Customer from order Action -Delete Update Action.

Table 6.5 Evaluation test case for extensibility

	New Action						
Element	**Retrieve**	**OnHold**	**Resume**	**Partial**	**Combine**	**Split**	**Transfer**
*Order (root)	X	X	X	X	X	X	X
ID	X	X	X	X	X	X	X
IssueDate							
BuyerCustomer-Party							
SellerSupplier-Party							
OrderLine							
Accepted-Indicator							

Table 6.6 New Actions for Order document with required elements

Listing 6.1 New extension action "Retrieve"

```
<order ID="0001" Action="Retrieve">
</order>
```

An order can be retrieved using the new action "Retrieve" to obtain the latest updated purchase order. This can be used to verify that a purchase order request is being processed correctly.

Some other potential actions can be added to the schema to extend the procurement functionality such as: "OnHold" and "Resume".

Listing 6.2 New extension action "OnHold"

```
<order ID="0001" Action="OnHold">
</order>
```

Listing 6.3 New extension action "Resume"

```
<order ID="0001" Action="Resume">
</order>
```

A service provider can provide a function of delaying of an order by specifying in the requesting message with the action “OnHold”. This will put the order on hold for a period of time then the action “Resume” will continue to process the purchase order following a later request.

Listing 6.4 New extension action “Partial”

```
<order ID="0001" Action="Partial">
</order>
```

An action of “Partial” indicates that the order can be fulfilled partially and can be delivered separately. This is particularly useful when some items are in back order, and as a result the current stocks can be dispatched quickly.

Listing 6.5 New extension action “Combine”

```
<order ID="0001" Action="Combine">
</order>
<order ID="0002" Action="Combine">
</order>
```

An action of “Combine” indicates that multiple orders can be combined and dispatched together.

Listing 6.6 New extension action “Split”

```
<order ID="0001" Action="Split">
</order>
```

An action of “Split” indicates that the order should split out to separate orders and can be delivered separately; this is particularly useful when some items are dispatched from multiple locations.

Listing 6.7 New extension action “Transfer”

```
<orderLine ID="0002" Action="Transfer">
</orderLine>
```

An action of “Transfer” indicates that the items are located in a different warehouse and required a transfer for completion.

6.3.5 FLEXIBILITY

A “changePurchaseOrderRequest” in the conventional UBL approach will require sending an updated purchase order (i.e. a new version of the purchase order). The service provider will need to identify the changes (i.e. the differences between the updated order and the previous purchase order request) then processes the items accordingly. The task is even more complicated if the purchase order is currently in the process of being handled because the provider will need to reconcile the original request with the changePurchaseOrder to identify the discrepancies and fulfil the changes necessary.

In contrast to the proposed minimalist approach, the update request will only contain the items that need to be changed by using the indicated actions on the purchase order items. Furthermore, multiple actions can be incorporated into a single request message hence the service provider will only require to process the discrepancies. An operation can process multiple actions corresponding to a business document, i.e. ability to perform “cancel” and “change” any “OrderLine” within an “order” in just one message. This feature not only can reduce the size of the message, but also indicates clearly what changes need to be processed. The service provider can process the message quickly and effectively.

The following sample demonstrating the action “Update” and “Cancel” can be performed by a specific OrderLine within an Order. This is unlike the traditional UBL approach of change Order where a new version of ChangeOrder needs to be sent or Cancel existing Order and then resend a complete new Order.

Test Case	Case Study Description
cancel and/or update any orderline	-Test ProcessOrder(order.xml) OrderLine - cancel and update any orderline - cancel or update any orderline

Table 6.7 Evaluation test case for flexibility

Listing 6.8 Multiple actions in a message

```
<order ID="0001" Action="Update">
<ChangeOrder>
  <OrderLine Action="Update">
    <OrderLineNumber>01</OrderLineNumber>
    <Item>
          <ItemID>002</ItemID>
          <Description>Pen</Description>
          <Price>9.99</Price>
    </Item>
    <Quantity>1</Quantity>
  </OrderLine>
<OrderLine Action="Cancel">
    <OrderLineNumber>02</OrderLineNumber>
    <Item>
          <ItemID>001</ItemID>
           <Description>Book</Description>
           <Price>99.99</Price>
    </Item>
    <Quantity>1</Quantity>
  </OrderLine>
</ChangeOrder>
</order>
```

6.3.6 MAINTAINABILITY

There are many business documents as well as interfaces in any domain-specific e-business applications; hence managing and frequently updating them can become difficult. This approach can help reduce the number of documents and interfaces for easy management, i.e. developers need only manage one document instead various business process documents.

In the conventional UBL approach, each operation will have a request and a corresponding response message. For instance, a typical purchase order scenario will have four operations and five business documents to maintain. In contrast the proposed approach will require only two operations and a single business document (Table 6.8).

Business Process	**Traditional UBL operation**	**Proposed Operation**
New Order	PurchaseOrder(OrderRequest.xml)	ProcessOrder(Order.xml)
Update Order	ChangeOrder(OrderChange.xml)	
Cancel Order	CancelOrder(OrderCancel.xml)	
Response Order	ResponseOrder(OrderResponse.xml) ResponseOrder(OrderResponseSimple.xml)	ResponseOrder(Order.xml)

Table 6.8 Traditional UBL operation and Proposed operation

Test Case	**Case Study Description**
Fewer documents to maintain.	-Compare with UBL's operation and XSD schemas.

Table 6.9 Evaluation test case for maintainability

6.3.7 COMPARISON OF THE OUTCOMES

Feature	UBL Services	Proposed Services
Minimise payload size	Resend the update order	Resend the update item
Granularity	Coarse-Grained - Send complete order document	Multi-Grained - Send single or more item
Reusability	Reuse based on defined coarse-grained operation	Higher reuse due to multi-grained operation
Extensibility	Create new operation and new document	Add new action and element to the existing interface
Flexibility	Specific business process operation	Allow single or multiple relevant actions in an operation
Maintainability	A set of operations and corresponding business documents	Single interface per business document

Table 6.10 Comparison between UBL services and proposed services

6.4 Discussion

The method exposes a minimalist API and then encapsulates it to add the relevant functionality. This way the developer can easily separate the derived behaviour (based on the business document related functionality, i.e. new PurchaseOrder, UpdatePurchaseOrder, or CancelPurchaseOrder) from the implementation-specific (minimalist) behaviour (based on the business document, i.e. purchaseOrder). Implementing a new minimalist version of an interface will give all clients access to the derived behaviour interface to the back-end. The proposed method can be applicable to any case study that has multiple business events for a business document. This unified message pattern contains the attribute “Action”, which defines “What to do” with its

element. This not only provides a clear instruction to a specific element but also allows multiple actions to be incorporated.

Based on the Service Principle outlined in Section 6.1, we have addressed and evaluated every feature. Firstly, reusability is addressed based on the generic operation and elements as well as multi-grained operation (both fine-grained and coarse-grained messages). It is reusable due to the fine-grained logic operation being repeatedly reused in a different business process. Secondly, extensibility is addressed by the notion of the Command pattern interface, which makes it possible to add or remove a command dynamically without changing the existing interface. The interface is also more flexible due to one or more commands being put into the message, which lets the operation be invoked in runtime. Abstraction principle is achieved by exposure through a minimal interface. Finally, the message size is reduced since it only exposes or sends the required data. This also reduces maintenance due to fewer interfaces and document schemas.

Although there are some disadvantages in employing this method, developers can learn to understand the schemas as well as the interfaces before integrating elements into their web application. Developers need to re-develop existing applications to ensure messages are compatible to the new interfaces. However, the ability to process multiple actions and reduce redundancy provides a better way for communicating between multiple services, especially in domain-specific e-business applications where documents are large and complex. Furthermore, the composition of Web Services is now simplified since fewer interfaces are needed to start with. Another important feature is extensibility, where changing the business document will no longer affect the composition pattern because the client is more stable. More importantly, the developer maintains a reduced number of business documents for multiple corresponding business processes.

CHAPTER 7
CONCLUSION

7.1 Research Summary

The proposed Web Service interface design addresses the problem of reusability and extensibility by adopting the Command pattern and generic minimalist approach to designing the messages (see Chapter 4). The proposed method reduces the large volumes and size of global trade domain service messages being exchanged. This was analysed and evaluated in Chapter 6. In the traditional UBL approach, updating an order requires sending a new version of the document. This is explained by the traditional e-business practice of exchanging the complete document (i.e. the document becomes a form or a contractual agreement).

If we only change an order item, do we really need to send an entire document to the business partner? If the majority of the information was previously sent and is redundant, we argue that sending the required and necessary information is enough to justify the service invocation call. From there we have arrived at the concept of the minimalist interface design where anything that is not important will become optional or redundant. The document itself will become part of the processing interface where instructions are stored in the document. We contend that any Web Service interface that has multiple operations to process a related business document should be abstract (i.e. become a generic operation). Here this operation will process the document according to the instruction given to the element.

The main goal of this thesis is to develop a Web Service interface design method for e-business application such that its interface is reusable, flexible and extensible. We begin by investigating the existing e-business document standards, and Web Service interface and composition designs. These standard documents all promote interoperability by adopting the Core Component-based elements. Some business documents, such as UBL and xCBL do adopt the Code Values or Action Types to a modification request message to indicate to the receiver how to process these intermittent elements. These are some examples of the Command pattern where instructions are passed as parameters for processing. However, use of these Command patterns is limited to the "modify request" rather than throughout the messages. Furthermore, these approaches still require multiple documents/schema for order, cancelOrder, changeOrder and responseOrder.

The change code lists in xCBL are long and complicated, and to change the date of an order delivery, we need to send the code "ChangeOfDates". To update an order in UBL and GS1, an updated revised order will need to be sent to overwrite the existing order. By analysing these business documents, we learn the flexibility and extendibility of these approaches. Flexibility is due to the use of multiple actions/commands, which can be sent and processed. Extendibility is due to the new actions, which can be added and removed. Developers only need to do minimal or no changes to their invocation calls to adapt to the new updated services.

We realise that by placing the processing instruction in the service elements, every object can be implemented with different actions. If we treat the document as an object and this document contains a collection of business objects (i.e. purchaseOrder as a business document), then changing an order item means sending an updated line item accordingly. We allow the exposed operations through the business document rather than as a separate business event. Therefore, instead of advertising a Web Service interface with these operations on the document, this Web Service interface can now have any operations functioning through the schema action elements. Studies and reviews of currently available methodologies for Web Service interface design revealed that limited attention has been paid to the design. Most players in the industry tend to adopt coarse-grained business processes to minimise Web Service invocation, whilst

others choose fine-grained business processes to maximise reusability. This is due to the convention of interoperability and interchanging between business partners as mentioned before.

These business processes and documents will be standardised and governed by a domain-specific organisation such as UBL, OTA and UN/EDIFACT. However, these approaches are complex and inefficient due to the large number of business processes and documents required. These business documents and Web Service interfaces also require a considerable amount of time and effort to standardise. Reusability is based on coarse-grained operation with business documents, which is hardly scalable due to its limitation. Extensibility is achieved by releasing a newer version of the standardised business document. Flexibility and efficiency are negligible and are not addressed in the current methodology. Therefore, a well-designed Web Service interface is a key requirement for ensuring a high level of interoperability and effectiveness in complex e-business applications.

Our proposed method addresses the limitations based on combining the command interface style and the flexible Generic Web Service interface to form a unified minimalist interface. The redefined business document takes interoperability into account by adopting the existing elements as well as the structure from UBL's Core Components. We are only introducing the "action" attribute to the appropriate elements. This will have minimal impact on reusability since all service operations are stateless and the sequences of the messages are governed by the unique identifier. Thus, communicating with existing services can be done by employing a process-switching layer.

The minimalist design approach can reduce the number of interfaces to their simplest form, thus further simplifying the interaction outcome. It is more stable due to the similarity to the REST style interface where processing instructions are defined in the message (i.e. ProcessOrder can handle purchaseOrder, cancelOrder and updateOrder). This enables a single operation to contain multiple business events, which can replace three or more operations where each operation represents a single business event. Hence

the operations do not have to change often when updates are required. It is more flexible due to the Generic Web Service where operations are defined in the message, relaxing the need for traditional operational methods.

This approach can transform a traditional business document interface into a more reusable, flexible, extensible and efficient interface. In particular, it is most effective for those interfaces that have multiple business events for a business document. We introduce a process-switching layer in the operation and XML schema in order to define the "what to do" actions. Simplifying and abstracting away unnecessary elements and operations is a minimalist strategy that unifies the interface.

7.2 Research Contributions

Our approach has confirmed the Web Service design principles as outlined in Section 6.1. We have also designed and implemented a prototype in order to evaluate the effectiveness of the proposed methodology. According to the thesis objectives as defined in Chapter 1 (Section 1.10), the major contributions of this research are summarised below:

- We have verified and satisfied Goal 1, showing that this approach is reusable due to its generic multi-grained operation. Fine-grained operation can be reused in coarse-grained messages while both fine-grained and coarse-grained messages can be composed in different Web Services to form more complex e-business applications. The interface can be reused for invoking multiple actions as well as other service calls to a "PurchaseOrder" business entity.

- One or more "change" actions in a multiple fine-grained order line item can be composed to a coarse-grained update "PurchaseOrder" document, or multiple fine-grained "New" order line items. This can be composed as a coarse-grained "New" "PurchaseOrder" document. Cancellation of an order is achieved by sending a fine-grained action "Cancel" to "PurchaseOrder" document. This addresses the requirements for Goal 2.

- Extensibility is also verified in this approach since new functions can be added or removed without affecting the existing interface; new actions can be defined or removed in the schema. New actions and required elements can be put into the messages and these allow the operation to be invoked at runtime. This is an important feature since Web Services are designed to be composed; if changing the interface might mean breaking up the composition, then it is not a feasible approach.

- This approach is verified by comparing it to the flexible requirements as defined in Goal 3. Since we are only exposing or sending the required data, fewer interfaces can mean less maintenance. This will also lead to a reduction in interaction calls since multiple fine-grained actions can be invoked on the service. The ability to process multiple actions and reduce redundancy provides a better and more efficient approach in communicating between e-business applications where documents are large and complex.

- We have also addressed Goal 4 by designing, implementing a prototype and evaluating the proposed method, which does indeed follow the design principles and is superior to existing design methods. In particular, some cases might provide up to 70% less processing time and payload size for the same output result.

7.3 Future Work

Based on the findings in this study and in particular the limitations observed in Chapter 6, future research work should address the following issues with consideration of the dynamic and evolving nature of the e-business industry.

Firstly, we are required to study composition in relation to the proposed methodology. Since the proposed interfaces design method differs from the traditional approach, current composition methodology might not be applicable to this approach or may not produce a satisfactory result. New composition methodologies can make use of the command pattern interface and take advantage of the flexibility feature.

Secondly, we can apply this approach to other additional vertical domains such as OTA travel booking, in order to verify cross-domain application. For instance, we can verify that a flight booking service can be composed of a duty-free purchase order service for additional purchasing.

Thirdly and finally, although our application can process UBL documents, we need to study the impact of processing the transformed UBL document such as those existing applications that are still composed using the traditional UBL approach. Since it is possible that developers may require modifying the existing interface to communicate with the proposed interface, it is necessary to investigate a more feasible method during the transition.

REFERENCES

Arenas, M. 2006, 'Normalization Theory for XML', *Sigmod Record*, vol. 35, no. 4, pp. 57-64.

Arsanjani, A. et al. 2008, 'SOMA: A method for developing service-oriented solutions', *IBM Systems Journal*, vol.47, no. 3, pp. 377–396.

Artus, D. 2006, 'SOA realization: Service design principles – Service design to enable IT flexibility/', viewed 13/08/2010, <http://www.ibm.com/developerworks/webservices/library/ws-soa-design/>.

Baghdadi, Y. 2004, 'A business model for B2B integration through Web Services', Proceedings of the IEEE International Conference on E-Commerce Technology, pp. 187-194.

Baghdadi, Y. 2005, 'A Business model for deploying Web Services: A data centric approach based on factual dependencies', *Inf. Syst. E-Business Management*, vol. 3, no. 2, pp. 151-173.

Barros, A., Dumas, M., and Oaks, P. 2005, 'A critical overview of the Web Services choreography description language', *BPTrends Newsletter*, 3, pp. 1-24.

Bean, J. 2009, *SOA and Web Services Interface Design, principles, techniques and standards*, The Morgan Kaufmann/OMG Press.

Beyer, D. Chakrabarti, A and Henzinger, T. 2005, 'Web Service Interfaces'. In Proceedings of the 14th International World Wide Web Conference (WWW 2005). ACM. New York, pp. 148 -159.

Berardi, D., Calvanese, D.,De Giacomo, G., Lenzerini, M,. and Mecella, M. 2003, 'Automatic composition of e-services that export their behaviour'. Proceedings of 1st International Conference on Service-Oriented Computing (ICSOC), LNCS. Trento, Italy, pp. 43-58.

Bloomberg, J. 2006, 'Should All Services be reusable?', viewed 2/5/2011, <http://www.zapthink.com/report.html?id=ZAPFLASH-2006531>.

Brogi, A., C. Canal et al. 2004, 'Formalizing Web Services Choreographies', 1st International Workshop on Web Services and Formal Methods (WS-FM 2004), Pisa, Italy, pp. 73-94.

Business Object Document Architecture, OAGIS Release 9.2, viewed 3/4/2007, <http://orgis.org>.

Casati, F., Castano, S., Fugini, M.G., Mirbel-Sanchez, I., Pernici, B. 2000, 'Using Patterns to design rules in workflows', *IEEE Transactions on Software Engineering*, vol. 26, no. 8, pp. 760-784.

Christian, H. 2005, 'UMM - Consistent B2B Analysis & Design on Top of ebXML and Web Services', Proceedings of the IEEE International Conference on e-Technology, e-Commerce and e-Service 2005 (EEE-05), IEEE Computer Society, Hong Kong (China). pp. 34-42.

Curbera, F., Goland, Y. et al. 2002, 'Business Process Execution Language for Web Services, Version 1.0', viewed 3/5/2007, <http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.pdf >.

Curbera F., Nagy, W. and Weerawaran, S. 2001, 'Web Services, Why and How'. Workshop on Object Oriented Web Services OOPSLA Tampa, Florida, USA.

Dan, A., Johnson, D. and Carrato, T. 2008, 'SOA Service Reuse by Design', Proceedings of the 2nd International Workshop on Systems Development in SOA Environments, Leipzig, Germany, pp. 25-28.

Dirk, B, A. C. and Henzinger, T. A. 2005, 'Web Service Interfaces', WWW '05 Proceedings of the 14th international conference on World Wide Web. ACM, New York, pp. 148-159.

ebXML Core Components specification, 2010, viewed 11/05/2011, <http://www.unece.org/cefact/ebxml/CCTS_V2-01_Final.pdf>.

Erl, T. 2007, 'The principles of Service-Oriented: Principle interrelationships and service layers', viewed 03/04/2008, <http://searchsoa.techtarget.com/tip/The-principles-of-service-orientation-part-6-of-6-Principle-interrelationships-and-service-layers>.

Erl, T. 2008, *SOA Principles of Service Design*, Prentice Hall/PearsonPTR, Hardcover, available at <http://www.soaprinciples.com/p6.asp>.

Feuerlicht, G., Lozina, J. 2007, 'Understanding Service Reusability'. In 15th International Conference Systems Integration 2007. VSE Prague, Prague, pp. 144-150.

Feuerlicht, G. and Meesathit, S. 2004, 'Design Method for Interoperable Web Services', International Conference on Service-Oriented Computing, New York, USA, November 2004. In Proceedings of the 2nd International Conference on Service-Oriented Computing, ed. Aiello, M et al. ACM, New York, pp. 299-307.

Feuerlicht, G. and Meesathit, S. 2004, 'Design Framework for domain-specific service interface', Workshop on Web Services: Modeling, Architecture and Infrastructure, Porto, Portugal, August 2004. In Proceedings of the 2nd International Workshop on Web Services: Modeling, Architecture and Infrastructure WSMAI 2004, ed. Bevinakoppa, S. and Hu, J. INSTICC Press, Porto, Portugal, pp. 109-115.

Fielding, R.T. 2000, 'Architectural styles and the design of network-based software architectures', Doctoral Dissertation, University of California, Irvine.

Fu, X., T. Bultan, et al. 2004, 'WSAT: A tool for Formal Analysis of Web Services', 16th International Conference on Computer Aided Verification (CAV), Boston, MA., pp. 510-514.

Gardner, T. 2003, 'UML Modelling of Automated Business Process with Mapping to BPEL4WS', European Workshop on Object Orientation and Web Services, Darmstadt, Germany.

Gemma, E., 'Design Patterns: Elements of Reusable Object Oriented Software' viewed 01/05/2011, <http://www.vico.org/pages/PatronsDisseny/Pattern%20Command/>.

Glushko, J. and McGrath, T. 2005, 'Document Engineering: analyzing and designing the semantics of Business Service Networks', Proceedings of the IEEE EEE05 International Workshop on Business Services Networks, vol. 87. IEEE Press, Piscataway, New Jersey, pp. 2-2

Glushko, J. and McGrath, T. 2005, 'Document Engineering: Analyzing and Designing Documents for Business Informatics and Web Services', *Journal of Documentation*, vol. 63, no. 2. pp. 288-290.

Gordijn, J., Akkermans, J.M., and van Vliet, J.C. 2000, 'Business modelling is not process modelling'. In Mayr, H.C., Liddle, S.W., and Thalheim, B. (eds.), *ER Workshops 2000*. LNCS, vol. 1921. Springer, Heidelberg, pp. 40-51.

Gottschalk, K., Graham S., Kreger, H. and Snell, J. 2002, 'Introduction to Web Services Architecture', *IBM Systems Journal*, vol. 41, no. 2. pp. 170-177.

Hanson, J. 2003, 'Coarse-grained interfaces enable service composition in SOA', viewed 01/10/2010, <http://builder.com.com/5100-6386-5064520.html>.

Henkel, M and Jelena, Z. 2005, 'Approaches to Service Interface Design', Proceedings of the Web Service Interoperability Workshop, First International Conference on Interoperability of Enterprise Software and Applications.

Henkel, M., Johannesson, P., Perjons, E., and Zdravkovic, J. 2007, 'Value and Goal Driven Design of E-Services'. In Proceedings of the IEEE International Conference on E-Business Engineering (Icebe 2007). IEEE Computer Society, Washington, pp. 295-303.

Hinkelman, S, Buddenbaum, D and Zhang, L. 2006, 'Emerging patterns in the use of XML for information modelling in vertical industries', *IBM System Journal*, vol. 45, no. 2, pp. 373-388.

Hiroshi, M. 2002, 'New trends in e-business: from B2B to Web Services', *New Generation Computing*, vol. 20, no. 1, pp.125-139.

Hong, S. and Feurerlicht, G. 2008, 'Web Service Interface Design for e-Business Application: A Minimalistic Design Approach', Proceeding NWESP '08 Proceedings of the 2008 4th International Conference on Next Generation Web Services Practices, Seoul, South Korea, pp. 143-150.

Hogg, K., Chilcott, P., Nolan, M. and Srinivasan, B. 2004, 'An Evaluation of Web Services in the design of a B2B Application', ACSC '04: Proceedings of the 27th Australasian Conference on Computer Science – 2004, vol. 26, Australian Computing Society. Darlinghurst, Australia, pp. 331-340.

Huang, Y. and Chung, Y. 2003, 'A Web Services-Based Framework for Business Integration Solutions', *Journal of Electronic Commerce Research and Application*, vol. 2, no. 1, pp. 15-26.

Hui, M. T., Wan M.N. and Wan, K. 2006, 'A Comparative Study of Interface Design Approaches for Service-Oriented Software', APSEC, 13th Asia Pacific Software Engineering Conference (APSEC'06). Kanpur, India, pp. 147-156.

Jakub, R. 2006, 'Performance Implications of Design Pattern Usage in Distributed Applications Case Studies in J2EE and .NET'. Proceedings of the ISSTA 2006 workshop on Role of Software Architecture for Testing and Analysis. ACM, 2006, pp. 1-11.

Jason, B. 2006, 'Should All Services be reusable', viewed 12/06/2006, <http://www.zapthink.com/report.html?id=ZAPFLASH-2006531>.

Java Blueprint 2004, 'Designing Web Service with J2EE – Managing complex Web Service interactions', viewed 11/05/2008,

<http://java.sun.com/blueprints/guidelines/designing_webservices/html/architecture6.html>.

Jeff, H. 2006, 'Coarse-grained interfaces enable service composition in SOA', viewed 11/07/2008, <http://builder.com.com/5100-6386-5064520.html>.

Jianwen, S. 2005, 'Web Service Interactions: Analysis and Design', The Fifth International Conference on Computer and Information Technology. IEEE, Shanghai, pp. 21-23.

Juric, M.B. 2006, 'BPEL: Service composition for SOA', viewed 15/05/2010, <http://www.javaworld.com/javaworld/jw-07-2006/jw-0710-bpel.html?lsrc=jwrss>.

Kabak, Y. and Dogac, A. 2010, 'A survey and analysis of electronic business document standards', *ACM Computing Surveys*, vol. 42, no. 3, article 11.

Kaminski, P. Litoiu, M. and Müller, H. 2006, 'A design technique for evolving Web Services'. In Proceedings of the 2006 Conference of the Center for Advanced Studies on Collaborative Research (CASCON '06), ed. Hakan Erdogmus, Eleni Stroulia, and Darlene Stewart. IBM Corp., Riverton, New Jersey, article 23.

Khalaf, R. 2006, 'From RosettaNet PIPs to BPEL processes: A three level approach for business protocols', *Data & Knowledge Engineering*, vol. 61, no. 1, pp. 23-28.

Lau, D. and Mylopoulos, J. 2004, 'Designing Web Services with Tropos'. Proceedings of the IEEE International Conference on Web Services (ICWS'04). pp. 306.

Legner, C. 2007, 'Design Principles for B2B Services – An Evaluation of two alternative service design', In Proceeding of Services Computing, SCC. IEEE International Conference. IEEE, Salt Lake City, pp. 372-379.

Lemanhieu, Wilfried, et al. 2003, 'An Event Based Approach to Web Service Design and Interaction', *Web Technologies and Applications, Lecture Notes in Computer Science*, vol. 2642, pp. 333-340.

Leymann, F. and Roller, D. 2002, 'Business process in a Web Service world - A quick overview of BPEL4WS', viewed 02/03/2008, <http://www-128.ibm.com/developerworks/library/ws-bpelwp/>.

Limthanmaphon. B. and Zhang, Y. 2003, 'Web Service Composition with case based reasoning'. In Proceedings of the Fourteen Australian Database Conference (ADC2003), vol. 17. Australian Computer Society, Darlinghurst, Australia, pp. 201-208.

Li, L. and Wu, C. 2005, 'Two-way Web Service: From Interface Design to interface Verification'. Proceedings of the 2005 IEEE International Conference on Web Services, IEEE, pp. 525-532.

Martin, H. and Jelena Z. 2005, 'Approach to Service Interface Design'. In Proceedings of the Web Service Interoperability Workshop, First International Conference on Interoperability of Enterprise Software and Applications (INTEROP-ESA'2005), Hermes Science Publisher, Geneva, Switzerland, February 22 - 25, Hermes Science Publisher.

McCarthy, J. 2002, 'Reap the benefits of Document Style Web Services – Web Services are not exclusively designed for handling remote procedure calls', IBM Developer Works, SOA and Web Services.

McManus, E. 2011, 'Java API Design Guidelines', viewed 15/07/2011, <http://www.artima.com/weblogs/viewpost.jsp?thread=142428>.

Medicke, J. 2011, 'Future-Proofing Solution with Coarse-Grained Service-Oriented Architecture', viewed 15/09/2011, <http://soa.sys-con.com/read/39848.htm>.

Meesathit, S. 2005, 'Web Services design in context of domain specific e-business', PhD thesis, University of Technology, Sydney.

Mecella, M. and Pernici, B. 2000, 'Designing Components for e-Services', Proceedings of the VLDB Workshop on Technologies for e-Services (VLDB-TES2000), Cairo, Egypt. pp. 177-187.

Melaika, S, Nelin, C, Qu, R, and WolfSon, D. 2002, 'DB2 Web Services' *IBM Systems Journal*, vol. 41, no. 2, pp. 666-685.

Milanovic, N. and Malek, M. 2004, 'Current solutions for Web Service composition', *Internet Computing, IEEE* , vol. 8, no. 6, pp. 51-59.

Millard, D.E., Howard, Y., Abbas, N., Davis, H.C., Gilbert, L., Wills, G.B., and Walters, R.J. 2009, 'Pragmatic Web Service design: An agile approach with the service responsibility and interaction design method', *Computer Science - Research and Development*, vol. 24, no. 4, pp. 173-184.

Morgan, G. 2011, 'Design for Flexibility', viewed 01/10/2011, <http://blogs.msdn.com/gabriel_morgan/archive/2006/10/03/Design-for-Flexibility.aspx>.

Nayak, N., Linehan, M. et al. 2007, 'Core business architecture for a service-oriented enterprise', *IBM Systems Journal*, vol. 46, no. 4, pp. 723–742.

OASIS, 'Web Services Atomic Transaction (WS-AtomicTransaction)' Version 1.1, viewed 1/03/2014, <http://docs.oasis-open.org/ws-tx/wstx-wsat-1.1-spec-errata-os/wstx-wsat-1.1-spec-errata-os.html>.

Open Application Group, viewed 10/04/2010, <http://openapplications.org>.

Open Travel Alliance, viewed 11/05/2011, <http://opentravel.org>.

Patterson, L, 2006, *Advanced ActionScript 3 with Design Patterns*, Pearson Education,.

Papazoglou, M. and Georgakopoulos, D. 2003, 'Service-Oriented Computing'. *ACM Press*, vol. 46, no. 10, pp. 99-101.

Papazoglou, M. and Yang, J. 2002, 'Design Methodology for Web Services and Business Processes'. In TES '02: Proceedings of the Third International Workshop on Technologies for E-Services. Springer-Verlag, London, pp. 54-64.

Provost, W. 2002, 'Normalizing XML, Part 1', viewed 1/10/2011, <http://www.xml.com/pub/a/2002/11/13/normalizing.html>.

Radeka, K. 2003, 'Designing a Web Services Project for Maximum Value: The 90 Day Challenge', OOPSLA 2002 Practitioners Reports. ACM, New York, p.1.

Rodriguez, A. 2008, 'RESTful Web Services: The basics', IBM developerWorks, viewed 15/03/2014, <https://www.ibm.com/developerworks/webservices/library/ws-restful/>.

Ronald, S. 2011, 'Solving the Service Granularity Challenge', viewed 1/10/2011, <http://searchwebservices.techtarget.com/tip/1,289483,sid26_gci1172330,00.html>.

Ruiz, M. and Pelechano, V. 2006, 'Designing Web Services for Supporting User Tasks: A Model Driven Approach Advances in Conceptual Modeling', *Advances in Conceptual Modeling - Theory and Practice*, vol. 4231, pp. 193-202.

Ruiz, M. and Pelechano, V. 2007, 'Model Driven Design of Web Service Operations using Web Engineering Practices'. *Emerging Web Services Technology*, ed. Calisti, M, Walliser, M. et al. Whitestein Series in Software Agent Technologies and Autonomic Computing, Springer, pp. 83–100.

Stanley, Y. and Li, H. 2001, 'Business Object Modeling, validation and Mediation for Integrating Heterogeneous Application Systems', *Journal of System Integration*, vol. 10, pp. 307-328.

Schmit, B.A. and Dustdar, S. 2005, 'Towards Transactional Web Services'. 1st IEEE International Workshop on Service-oriented Solutions for Cooperative Organizations (SoS4CO '05), co-located with the 7th International IEEE Conference on E-Commerce Technology (CEC 2005), Munich, Germany. pp. 12-20.

Stevens, M. 2002, 'Multi-Grained Services', viewed 01/10/2011, <http://www.developer.com/services/article.php/1142661>

Steve, V. 2005, 'Old Measures for New Services', *IEEE Internet Computing*, vol. 9, no. 6, pp. 72-74.

Skogan, D. 2004, 'Web Service Composition in UML'. In Proceedings of the 8th IEEE International Enterprise Distributed Object Computing Conference (EDOC 2004). pp. 47-57.

The Organization for the Advancement of Structured Information Standards (OASIS) Universal Business Language, viewed 10/06/2010, <https://www.oasis-open.org/committees/tc_home.php?wg_abbrev=ubl>.

The United Nations Centre for Trade Facilitation and Electronic Business, viewed 10/03/2011, <http://www.unece.org/cefact/>.

UN/EDIFACT is the international EDI standard maintained by the United Nations Centre for Trade Facilitation and Electronic Business, viewed 10/03/2011, <http://www.unece.org/cefact>.

WSDL and UDDI, 2005 viewed 10/03/2014, <http://www.w3schools.com/webservices/ws_wsdl_uddi.asp>.

Vadym, J. 2009, 'Generic Web Services - Extensible Functionality with Stable Interface'. In Proceedings of the 2009 IEEE International Conference on Web Services. IEEE, Los Angeles, pp. 1032-1034.

Vadym, J. 2009, 'Ensuring Service Backwards Compatibility with Generic Web Services'. In Proceedings of the 2009 ICSE Workshop on Principles of Engineering Service-Oriented Systems. IEEE Computer Society, Washington, D.C., pp. 95-98.

Van der, W. 2003, 'Don't go with the flow: Web Services composition standards exposed' *IEEE Intelligent Systems*, vol. 18, no. 1, pp. 72-76.

Vinoski, S. 2002, 'Putting the "Web" into Web Services: Web Service interaction models, Part 2'. *IEEE Internet Computing*, vol. 6, no. 4, pp. 90-92.

Wang, H, Huang, Z, Qu, Y. and Xie, J. 2004, 'Web Services: Problems and future directions', *Web Semantics: Science, Services and Agents on the World Wide Web*, vol. 1, no. 3, pp. 309-320.

Weigand, H., Johannesson P., Andersson B., and Bergholtz, M. 2009, 'Value-Based Service Modeling and Design: Toward a Unified View of Services'. In Proceedings of the 21st International Conference on Advanced Information Systems Engineering (CAiSE '09), ed. Pascal Eck, Jaap Gordijn, and Roel Wieringa. Springer-Verlag, Berlin, Heidelberg, pp. 410-424.

Weigand, H. et al. 2007, 'Strategic Analysis Using Value Modeling'–The c3-Value Approach. In: HICSS 2007, p. 175.

Woodman, S., D. Palmer et al. 2004, 'Notations for the Specification and Verification of Composite Web Services'. 8th IEEE International Enterprise Distributed Object Computing (EDOC) Conference, Monterey, California, pp. 35-46.

Zein, O.K. and Kermarrec, Y. 2006, 'Static/Semi-Dynamic and Dynamic Composition of Services in Distributed Systems', Telecommunications, 2006. AICT-ICIW '06. International Conference on Internet and Web Applications and Services, IEEE, p. 144.

APPENDICES

Appendix 1: Script to generate Database Table for case study

```
/*
The following scripts are used to generate the required database tables in order to run
the purchase order prototype application. There are four main tables: orders, orderLine,
customer and product.
*/

CREATE TABLE orders(
  ord_id int(10),
  ord_status varchar(7),
  ord_date varchar(7),
  ord_total int(10),
  cus_id int(10),
  PRIMARY KEY (ord_id),
  CONSTRAINT customer FOREIGN KEY (cus_id) REFERENCES customer (cus_id)
);

CREATE TABLE orderLine(
  orl_id int(10),
  orl_quantity int(10),
  pro_id int(10),
  ord_id int(10),
  PRIMARY KEY  (ord_id,orl_id),
  CONSTRAINT oders FOREIGN KEY (ord_id) REFERENCES orders (ord_id),
  CONSTRAINT product FOREIGN KEY (pro_id) REFERENCES product (pro_id)
);

CREATE TABLE customer(
  cus_id int(10),
  cus_name varchar(20),
```

```
  cus_address varchar(30),
  cus_contact int(10),
  PRIMARY KEY (cus_id)
);

CREATE TABLE product (
  pro_id int(10),
  pro_description varchar(25),
  pro_price int(10),
  PRIMARY KEY  (pro_id)
);
```

Appendix 2: Available Web Method as Web Service

```
/*
The following scripts are the available Web Service operations that are accessible via
Web Service WSDL, however, not all the method will be used. processOrderXML is
the main operation use for the prototype.
*/


@WebService(name = "MyWebService")
public interface MyWebService extends Remote {
   @WebMethod
   public void newOrder(Orders orders);
   @WebMethod
   public List<Customer> queryCustomerFindAll();
   @WebMethod
   public List<Orderline> queryOrderlineFindAll();
   @WebMethod
   public List<Orderline> queryOrderlineFindOrderlineById(int ord_id);
   @WebMethod
   public List<Orders> queryOrdersFindAll();
   @WebMethod
   public Orders queryOrdersFindOrdersById(int ord_id);
   @WebMethod
   public List<Product> queryProductFindAll();
   @WebMethod
   Product queryProductFindProductById(int pro_id) throws RemoteException;
   @WebMethod
   public void processOrderXML(String address);
   @WebMethod
   public void processOrder(Orders orders, String action);
   @WebMethod
   public void processOrderLine(Orderline orderline, String action);
   @WebMethod
   public void changeOrders(Orders orders);
   @WebMethod
   public void changeOrderLine(Orderline orderline);
}
```

Appendix 3: Persistence Unit for the Prototype Application

```
/*
The following is the persistence configuration in order to setup the required data
connection between the application and the database. This configuration define the
mapping between the entity object and the database table.
*/

<?xml version="1.0" encoding="UTF-8"?>
<persistence version="2.0" xmlns="http://java.sun.com/xml/ns/persistence"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://java.sun.com/xml/ns/persistence
http://java.sun.com/xml/ns/persistence/persistence_2_0.xsd">
  <persistence-unit name="OrderEntityPU" transaction-type="RESOURCE_LOCAL">
    <provider>org.eclipse.persistence.jpa.PersistenceProvider</provider>
    <class>Order.Entity.Customer</class>
    <class>Order.Entity.Order</class>
    <class>Order.Entity.OrderLine</class>
    <class>Order.Entity.Product</class>
    <properties>
      <property name="javax.persistence.jdbc.url"
value="jdbc:mysql://localhost:3306/Orders"/>
      <property name="javax.persistence.jdbc.driver" value="com.mysql.jdbc.Driver"/>
      <property name="javax.persistence.jdbc.user" value="root"/>
      <property name="javax.persistence.jdbc.password" value="root"/>
    </properties>
  </persistence-unit>
</persistence>
```

Appendix 4: Business Entities of the Prototype application

/*

The following is the Order object business entity class

*/

Order Entity:

```
package Order.Entity;

import java.io.Serializable;
import java.util.ArrayList;
import java.util.List;
import javax.persistence.CascadeType;
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.FetchType;
import javax.persistence.Id;
import javax.persistence.JoinColumn;
import javax.persistence.ManyToOne;
import javax.persistence.OneToMany;
import javax.persistence.Table;
import javax.persistence.Transient;

/**
 *This Order class is the representation of an order object and define the mapping in the
 *database table “orders”
 */
@Entity
@Table(name = "orders")
public class Order implements Serializable {
    private Integer id;
    private String createDate;
    private String status;
    private Integer total;
    private List<OrderLine> orderlines;
    private Customer customer;
    private String action;

/*
 * The default constructor of an Order object business entity
 */
    public Order() {
    }

/*
```

```
 * The Order object business entity constructor with creation_date, customer, status,
total and action
 */
   public Order(String create_date, Customer customer,
            String status, Integer total,String action) {
      this.customer = customer;
      this.createDate = create_date;
      this.customer = customer;
      this.status = status;
      this.total = total;
      this.action = action;
   }

   /**
    * Gets the value of the action of an order.
    */
   @Transient
   public String getAction() {
      return action;
   }

   /**
    * Set the value of the action of an order.
    */
   public void setAction(String action) {
      this.action = action;
   }

   /**
    * Gets the value of the createDate of an order.
    */
   @Column(name="create_date")
   public String getCreateDate() {
      return createDate;
   }

   /**
    * Set the value of the createDate of an order.
    */
   public void setCreateDate(String create_date) {
      this.createDate = create_date;
   }

   /**
    * Gets the value of the id of an order.
    */
   @Id
   @Column(name = "id")
```

```
    public Integer getId() {
        return id;
    }

    /**
     * Set the value of the id of an order.
     */
    public void setId(Integer id) {
        this.id = id;
    }

    /**
     * Gets the value of the status of an order.
     */
    @Column(name = "status")
    public String getStatus() {
        return status;
    }

    /**
     * Set the value of the status of an order.
     */
    public void setStatus(String status) {
        this.status = status;
    }

    /**
     * Gets the value of the total of an order.
     */
    @Column(name="total")
    public Integer getTotal() {
        return total;
    }

    /**
     * Set the value of the total of an order.
     */
    public void setTotal(Integer ord_total) {
        this.total = ord_total;
    }

    /**
     * Gets the list of orderLine of an order. Return empty list if not exist
     */
    @OneToMany(mappedBy = "order", cascade = {CascadeType.PERSIST,
CascadeType.REMOVE, CascadeType.MERGE}, fetch = FetchType.EAGER)
    public List<OrderLine> getOrderlines() {
        if(orderlines == null)
```

```
    {
        orderlines = new ArrayList<OrderLine>();
    }
    return orderlines;
}

/**
 * Set the list of orderLine of an order.
 */
public void setOrderlines(List<OrderLine> orderlines) {
    this.orderlines = orderlines;
}

/**
 * Add an orderLine to an order.
 */
public OrderLine addOrderline(OrderLine orderline) {
    getOrderlines().add(orderline);
    return orderline;
}

/**
 * Remove an orderLine from an order.
 */
public OrderLine removeOrderline(OrderLine orderline) {
    getOrderlines().remove(orderline);
    orderline.setOrder(null);
    return orderline;
}

 /**
 * Gets the customer of an order.
 */
@ManyToOne(fetch = FetchType.EAGER)
@JoinColumn(name = "customer_id", insertable = true, updatable = true)
public Customer getCustomer() {
    return customer;
}

/**
 * Set the customer of an order.
 */
public void setCustomer(Customer customer) {
    this.customer = customer;
}

/**
 * Gets the hashcode of an order.
```

```
     */
    @Override
    public int hashCode() {
        int hash = 0;
        hash += (id != null ? id.hashCode() : 0);
        return hash;
    }

    /**
     * Return true if an object is the same order.
     */
    @Override
    public boolean equals(Object object) {
        if (!(object instanceof Order)) {
            return false;
        }
        Order other = (Order) object;
        if ((this.id == null && other.id != null) || (this.id != null &&
!this.id.equals(other.id))) {
            return false;
        }
        return true;
    }

    /**
     * Return a String representation of an order object.
     */
    @Override
    public String toString() {
        return "Order.Entity.Order[id=" + id + "]";
    }
}
```

/*

The following is the customer object business entity class

*/

Customer Entity:

```
package Order.Entity;

import java.io.Serializable;
import java.util.ArrayList;
import java.util.List;
import javax.persistence.Column;
```

```
import javax.persistence.Entity;
import javax.persistence.Id;
import javax.persistence.NamedQueries;
import javax.persistence.NamedQuery;
import javax.persistence.OneToMany;
import javax.persistence.Table;
import javax.xml.bind.annotation.XmlTransient;

/**
 *This Customer class is the representation of a customer object and define the mapping
 * in the database table “customers”
 */
@Entity
@NamedQueries({
    @NamedQuery(name = "Customer.findAll", query = "select o from Customer o"),
    @NamedQuery(name = "Customer.findCustomerById", query = "select o from
Customer o where o.id = :id")
})
@Table(name = "customers")
public class Customer implements Serializable {

    @Id
    @Column(name = "id", nullable = false)
    private Integer id;
    @Column(name = "name")
    private String name;
    @Column(name = "address")
    private String address;
    @Column(name = "contact")
    private Integer contact;
    @OneToMany(mappedBy = "customer")
    private List<Order> orders;

    /**
     * Gets the value of the id of a customer.
     */
    public Integer getId() {
        return id;
    }

    /**
     * Set the value of the id of a customer.
     */
    public void setId(Integer cus_id) {
        this.id = cus_id;
    }

    /**
```

```
 * Gets the value of name of a customer.
 */
public String getName() {
   return name;
}

/**
 * Set the value of the name of a customer.
 */
public void setName(String cus_name) {
   this.name = cus_name;
}

/**
 * Gets the value of the address of a customer.
 */
public String getAddress() {
   return address;
}

/**
 * Set the value of the address of a customer.
 */
public void setAddress(String cus_address) {
   this.address = cus_address;
}

/**
 * Gets the value of the contact of a customer.
 */
public Integer getContact() {
   return contact;
}

/**
 * Set the value of the contact of a customer.
 */
public void setContact(Integer cus_contact) {
   this.contact = cus_contact;
}

/**
 * Gets the list of order of a customer. Return empty list if not exist
 */
@XmlTransient
public List<Order> getOrders() {
   if(orders == null)
   {
```

```
            orders = new ArrayList<Order>();
        }
        return orders;
    }

    /**
     * Set the list of orders to a customer.
     */
    public void setOrders(List<Order> ordersList) {
        this.orders = ordersList;
    }

    /**
     * Add an order to a customer.
     */
    public Order addOrder(Order order) {
        getOrders().add(order);
        order.setCustomer(this);
        return order;
    }

    /**
     * Remove an order from a customer.
     */
    public Order removeOrder(Order order) {
        getOrders().remove(order);
        order.setCustomer(null);
        return order;
    }

    /**
     * Gets the hashcode of a customer.
     */
    @Override
    public int hashCode() {
        int hash = 0;
        hash += (id != null ? id.hashCode() : 0);
        return hash;
    }

    /**
     * Return true if an object is the same customer.
     */
    @Override
    public boolean equals(Object object) {
        // TODO: Warning - this method won't work in the case the id fields are not set
        if (!(object instanceof Customer)) {
            return false;
```

```
        }
        Customer other = (Customer) object;
        if ((this.getId() == null && other.getId() != null)
                || (this.getId() != null && !this.getId().equals(other.getId()))) {
            return false;
        }
        return true;
    }

    /**
     * Return a String representation of a customer object.
     */
    @Override
    public String toString() {
        return "Edu.Order.Customer[id=" + this.getId() + "]";
    }
}
```

/*

The following is the OrderLine object business entity class

*/

OrderLine Entity:

```
/*
 * The following is the OrderLine object business entity class
 */
package Order.Entity;

import java.io.Serializable;
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.FetchType;
import javax.persistence.Id;
import javax.persistence.JoinColumn;
import javax.persistence.ManyToOne;
import javax.persistence.NamedQueries;
import javax.persistence.NamedQuery;
import javax.persistence.Table;
import javax.persistence.Transient;
import javax.xml.bind.annotation.XmlTransient;

/**
```

```
 *This OrderLine class is the representation of an orderline object and define the
 *mapping in the database table “orderlines”
 */
@Entity
@NamedQueries({
   @NamedQuery(name = "Orderline.findAll", query = "select o from OrderLine o"),
   @NamedQuery(name = "Orderline.findOrderlineById", query = "select o from
OrderLine o where o.id = :id")
})
@Table(name = "orderlines")
public class OrderLine implements Serializable {

   @Column(name = "ID")
   private Integer id;
   @Column(name = "quantity")
   private Integer quantity;
   @ManyToOne(fetch = FetchType.EAGER)
   @JoinColumn(name = "product_id")
   private Product product;
   @ManyToOne
   @JoinColumn(name = "order_id", insertable = true, updatable = true)
   private Order order;
   @Transient
   private String action;

   /*
    * The default constructor of an OrderLine object business entity
   */
   public OrderLine() {
   }

   * The OrderLine object business entity constructor with id, quantity, product, order
and action
 */
   public OrderLine(Integer id, Integer quantity,
         Product product, Order order, String action) {
      this.order = order;
      this.id = id;
      this.quantity = quantity;
      this.product = product;
      this.action = action;
   }

   /**
    * Gets the value of the action of an orderLine.
    */
   public String getAction() {
      return action;
```

```
}

/**
 * Set the value of the action of an orderLine.
 */
@Transient
public void setAction(String action) {
    this.action = action;
}

/**
 * Gets the value of the orderId of an orderLine.
 */
public Integer getOrder_Id() {
    return order.getId();
}

/**
 * Gets the value of the id of an orderLine.
 */
@Id
public Integer getId() {
    return id;
}

/**
 * Set the value of the id of an orderLine.
 */
public void setId(Integer id) {
    this.id = id;
}

/**
 * Gets the value of the quantity of an orderLine.
 */
public Integer getQuantity() {
    return quantity;
}

/**
 * Set the value of the quantity of an orderLine.
 */
public void setQuantity(Integer quantity) {
    this.quantity = quantity;
}

/**
 * Gets the Product of an orderLine.
```

```
 */
public Product getProduct() {
    return product;
}

/**
 * Set the Product of an orderLine.
 */
public void setProduct(Product product) {
    this.product = product;
}

/**
 * Gets the Order of an orderLine.
 */
@XmlTransient
public Order getOrder() {
    return order;
}

 /**
 * Set the Order of an orderLine.
 */
public void setOrder(Order order) {
    this.order = order;
    if (order != null) {
        this.id = order.getId();
    }
}

/**
 * Gets the hashcode of an orderLine.
 */
@Override
public int hashCode() {
    int hash = 0;
    hash += (id != null ? id.hashCode() : 0);
    return hash;
}

/**
 * Return true if an object is the same orderLine.
 */
@Override
public boolean equals(Object object) {
    if (!(object instanceof OrderLine)) {
        return false;
    }
```

```
        OrderLine other = (OrderLine) object;
        if ((this.id == null && other.id != null) || (this.id != null &&
!this.id.equals(other.id))) {
            return false;
        }
        return true;
    }

    /**
     * Return a String representation of an orderLine object.
     */
    @Override
    public String toString() {
        return "Order.Entity.OrderLine[id=" + id + "]";
    }
}
```

/*

The following is the Product object business entity class

*/

Product Entity:

```
package Order.Entity;

import java.io.Serializable;
import java.util.List;
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.Id;
import javax.persistence.NamedQueries;
import javax.persistence.NamedQuery;
import javax.persistence.OneToMany;
import javax.persistence.Table;
import javax.xml.bind.annotation.XmlTransient;

/**
 *This Product class is the representation of a product object and define the mapping in
 *the database table “products”
 */
@Entity
@NamedQueries({
  @NamedQuery(name = "Product.findAll", query = "select o from Product o"),
      @NamedQuery(name = "Product.findProductById", query = "select o from Product
o where o.id = :id")
```

```
})
@Table(name = "products")
public class Product implements Serializable {
  @Column(name="description")
   private String description;
   @Id
   @Column(name="id")
   private Integer id;
   @Column(name="price")
   private Integer price;
   @OneToMany(mappedBy = "product")
   private List<OrderLine> orderlines;

   /*
    * The default constructor of an Product object business entity
    */
   public Product() {
   }

  /* The Product object business entity constructor with descrition, id and price
*/
   public Product(String description, Integer id, Integer price) {
      this.description = description;
      this.id = id;
      this.price = price;
   }

   /**
    * Gets the value of the description of a product.
    */
   public String getDescription() {
      return description;
   }

   /**
    * Set the value of the description of a product.
    */
   public void setDescription(String description) {
      this.description = description;
   }

   /**
    * Gets the value of the id of a product.
    */
   public Integer getId() {
      return id;
   }
```

```
/**
 * Set the value of the id of a product.
 */
public void setId(Integer id) {
   this.id = id;
}

/**
 * Gets the value of the price of a product.
 */
public Integer getPrice() {
   return price;
}

/**
 * Set the value of the price of a product.
 */
public void setPrice(Integer price) {
   this.price = price;
}

/**
 * Gets the list of orderLine of a product.
 */
@XmlTransient
public List<OrderLine> getOrderlines() {
   return orderlines;
}

/**
 * Set the list of orderLine of a product.
 */
public void setOrderlines(List<OrderLine> orderlineList) {
   this.orderlines = orderlineList;
}

/**
 * Add an orderLine to an orderLine of a product.
 */
public OrderLine addOrderline(OrderLine orderline) {
   getOrderlines().add(orderline);
   orderline.setProduct(this);
   return orderline;
}

/**
 * Remove an orderLine to an orderLine of a product.
 */
```

```
    public OrderLine removeOrderline(OrderLine orderline) {
        getOrderlines().remove(orderline);
        orderline.setProduct(null);
        return orderline;
    }

    /**
     * Gets the hashcode of a product.
     */
    @Override
    public int hashCode() {
        int hash = 0;
        hash += (id != null ? id.hashCode() : 0);
        return hash;
    }

    /**
     * Return true if an object is the same product.
     */
    @Override
    public boolean equals(Object object) {
        if (!(object instanceof Product)) {
            return false;
        }
        Product other = (Product) object;
        if ((this.id == null && other.id != null) || (this.id != null &&
!this.id.equals(other.id))) {
            return false;
        }
        return true;
    }

    /**
     * Return a String representation of a product object.
     */
    @Override
    public String toString() {
        return "Order.Entity.Product[id=" + id + "]";
    }

}
```

Appendix 5: Data Access Object of the Prototype application

/*

The following is the Data Access object class, the interface to access the object

*/

Interface of the DAO :

```
package Order.DAO;

public interface IRepository<T> {
   public T merge(T entity);
   public T persist(T entity);
   public void remove(T entity);
   public T FindById(Integer id);
}
```

/*

The following is the Data Access object class, the interface to access the Order object

*/

Order DAO:

```
package Order.DAO;

import Order.Entity.Order;
import javax.persistence.EntityManager;
import javax.persistence.Query;

public class OrderDAO implements IRepository<Order> {

   /**
    * Constructor for OrderDAO.
    */
   public OrderDAO(EntityManager em) {
      this.em = em;
   }

   /**
    * Update an Order object.
    */
   public Order merge(Order order) {
      return em.merge(order);
   }
```

```
    /**
     * Persist an Order object.
     */
    public Order persist(Order order) {
        em.persist(order);
        return order;
    }

    /**
     * Remove an Order object.
     */
    public void remove(Order order) {
        order = em.merge(order);
        em.remove(order);
    }

    /**
     * Find Order object by Id.
     */
    public Order FindById(Integer id) {
        Query query = em.createQuery("select m from Order m where m.id = " + id);
        return (Order) query.getSingleResult();
    }
    private EntityManager em;
}
```

/*

The following is the Data Access object class, the interface to access the Product object

*/

Product DAO:

```
package Order.DAO;

import Order.Entity.Product;
import javax.persistence.EntityManager;
import javax.persistence.Query;

public class ProductDAO implements IRepository<Product> {

    /**
     * Constructor for ProductDAO.
     */
    public ProductDAO(EntityManager em) {
        this.em = em;
```

```
    }

    public Product merge(Product entity) {
        return em.merge(entity);
    }

    /**
     * Update a Product object.
     */
    public Product persist(Product entity) {
        em.persist(entity);
        return entity;
    }

    /**
     * Remove a Product object.
     */
    public void remove(Product entity) {
        entity = em.merge(entity);
        em.remove(entity);
    }

    /**
     * Find a Product object by Id.
     */
    public Product FindById(Integer id) {
        Query query = em.createQuery("select m from Product m where m.id = " + id);
        return (Product) query.getSingleResult();
    }
    private EntityManager em;
}


/*
The following is the Data Access object class, the interface to access the Customer
object
*/
```

Customer DAO:

```
package Order.DAO;

import Order.Entity.Customer;
import javax.persistence.EntityManager;
import javax.persistence.Query;
```

```
public class CustomerDAO implements IRepository<Customer> {

   /**
    * Constructor for CustomerDAO.
    */
   public CustomerDAO(EntityManager em) {
      this.em = em;
   }

   /**
    * Update a Customer object.
    */
   public Customer merge(Customer entity) {
      return em.merge(entity);
   }

   /**
    * Persist a Customer object.
    */
   public Customer persist(Customer entity) {
      em.persist(entity);
      return entity;
   }

   /**
    * Remove a Customer object.
    */
   public void remove(Customer entity) {
      entity = em.merge(entity);
      em.remove(entity);
   }
   private EntityManager em;

  /**
    * Find a Customer object by Id.
    */
   public Customer FindById(Integer id) {
      Query query = em.createQuery("select m from Customer m where m.id = " + id);
      return (Customer)query.getSingleResult();
   }
}
```

Appendix 6: Business Value Object of the Prototype application

```
/*
The following is the changeOrderType XML value object, it is a XML format of the
changeOrder object
*/
```

ChangeOrderType Value Object

```
package Order.Serializer;

import java.util.ArrayList;
import java.util.List;

import javax.xml.bind.annotation.XmlAccessType;
import javax.xml.bind.annotation.XmlAccessorType;
import javax.xml.bind.annotation.XmlElement;
import javax.xml.bind.annotation.XmlType;


/**
 * <p>Java class for ChangeOrderType complex type.
 *
 * <p>The following schema fragment specifies the expected content contained within
this class.
 *
 * <pre>
 * &lt;complexType name="ChangeOrderType">
 *   &lt;complexContent>
 *     &lt;restriction base="{http://www.w3.org/2001/XMLSchema}anyType">
 *       &lt;sequence>
 *         &lt;element name="OrderLine" type="{}OrderLineType"
maxOccurs="unbounded"/>
 *       &lt;/sequence>
 *     &lt;/restriction>
 *   &lt;/complexContent>
 * &lt;/complexType>
 * </pre>
 *
 */
@XmlAccessorType(XmlAccessType.FIELD)
@XmlType(name = "ChangeOrderType", propOrder = { "orderLine" })
public class ChangeOrderType {
```

```
    @XmlElement(name = "OrderLine", required = true)
    protected List<OrderLineType> orderLine;

    /**
     * Gets the value of the orderLine property.
     *
     * <p>
     * This accessor method returns a reference to the live list,
     * not a snapshot. Therefore any modification you make to the
     * returned list will be present inside the JAXB object.
     * This is why there is not a <CODE>set</CODE> method for the orderLine
property.
     *
     * <p>
     * For example, to add a new item, do as follows:
     * <pre>
     *    getOrderLine().add(newItem);
     * </pre>
     *
     *
     * <p>
     * Objects of the following type(s) are allowed in the list
     * {@link OrderLineType }
     *
     *
     */
    public List<OrderLineType> getOrderLine() {
        if (orderLine == null) {
            orderLine = new ArrayList<OrderLineType>();
        }
        return this.orderLine;
    }

}

/*
The following is the customerPartyType XML value object, it is a XML format of the
customerParty object
*/
```

CustomerPartyType Value Object:

```
package Order.Serializer;

import javax.xml.bind.annotation.XmlAccessType;
import javax.xml.bind.annotation.XmlAccessorType;
```

```
import javax.xml.bind.annotation.XmlElement;
import javax.xml.bind.annotation.XmlType;


/**
 * <p>Java class for CustomerPartyType complex type.
 *
 * <p>The following schema fragment specifies the expected content contained within
this class.
 *
 * <pre>
 * &lt;complexType name="CustomerPartyType">
 *   &lt;complexContent>
 *     &lt;restriction base="{http://www.w3.org/2001/XMLSchema}anyType">
 *       &lt;sequence>
 *         &lt;element name="ID" type="{http://www.w3.org/2001/XMLSchema}int"/>
 *         &lt;element name="Name"
type="{http://www.w3.org/2001/XMLSchema}string"/>
 *         &lt;element name="Address"
type="{http://www.w3.org/2001/XMLSchema}string"/>
 *         &lt;element name="Contact"
type="{http://www.w3.org/2001/XMLSchema}string"/>
 *       &lt;/sequence>
 *     &lt;/restriction>
 *   &lt;/complexContent>
 * &lt;/complexType>
 * </pre>
 *
 *
 */
@XmlAccessorType(XmlAccessType.FIELD)
@XmlType(name = "CustomerPartyType",
        propOrder = { "id", "name", "address", "contact" })
public class CustomerPartyType {

    @XmlElement(name = "ID")
    protected int id;
    @XmlElement(name = "Name", required = true)
    protected String name;
    @XmlElement(name = "Address", required = true)
    protected String address;
    @XmlElement(name = "Contact", required = true)
    protected String contact;

    /**
     * Gets the value of the id property.
     *
     */
```

```
public int getID() {
    return id;
}

/**
 * Sets the value of the id property.
 *
 */
public void setID(int value) {
    this.id = value;
}

/**
 * Gets the value of the name property.
 *
 * @return
 *     possible object is
 *     {@link String }
 *
 */
public String getName() {
    return name;
}

/**
 * Sets the value of the name property.
 *
 * @param value
 *     allowed object is
 *     {@link String }
 *
 */
public void setName(String value) {
    this.name = value;
}

/**
 * Gets the value of the address property.
 *
 * @return
 *     possible object is
 *     {@link String }
 *
 */
public String getAddress() {
    return address;
}
```

```
    /**
     * Sets the value of the address property.
     *
     * @param value
     *     allowed object is
     *     {@link String }
     *
     */
    public void setAddress(String value) {
        this.address = value;
    }

    /**
     * Gets the value of the contact property.
     *
     * @return
     *     possible object is
     *     {@link String }
     *
     */
    public String getContact() {
        return contact;
    }

    /**
     * Sets the value of the contact property.
     *
     * @param value
     *     allowed object is
     *     {@link String }
     *
     */
    public void setContact(String value) {
        this.contact = value;
    }
}
```

/*

The following is the itemType XML value object, it is a XML format of the itemType object

*/

ItemType Value Object:

```
package Order.Serializer;

import javax.xml.bind.annotation.XmlAccessType;
import javax.xml.bind.annotation.XmlAccessorType;
import javax.xml.bind.annotation.XmlElement;
import javax.xml.bind.annotation.XmlType;


/**
 * <p>Java class for ItemType complex type.
 *
 * <p>The following schema fragment specifies the expected content contained within this class.
 *
 * <pre>
 * &lt;complexType name="ItemType">
 *   &lt;complexContent>
 *     &lt;restriction base="{http://www.w3.org/2001/XMLSchema}anyType">
 *       &lt;sequence>
 *         &lt;element name="ProductId" type="{http://www.w3.org/2001/XMLSchema}int"/>
 *         &lt;element name="ProductName" type="{http://www.w3.org/2001/XMLSchema}string"/>
 *         &lt;element name="Price" type="{http://www.w3.org/2001/XMLSchema}string"/>
 *       &lt;/sequence>
 *     &lt;/restriction>
 *   &lt;/complexContent>
 * &lt;/complexType>
 * </pre>
 *
 *
 */
@XmlAccessorType(XmlAccessType.FIELD)
@XmlType(name = "ItemType",
        propOrder = { "productId", "productName", "price" })
public class ItemType {

    @XmlElement(name = "ProductId")
```

```
protected int productId;
@XmlElement(name = "ProductName", required = true)
protected String productName;
@XmlElement(name = "Price", required = true)
protected String price;

/**
 * Gets the value of the productId property.
 *
 */
public int getProductId() {
   return productId;
}

/**
 * Sets the value of the productId property.
 *
 */
public void setProductId(int value) {
   this.productId = value;
}

/**
 * Gets the value of the productName property.
 *
 * @return
 *     possible object is
 *     {@link String }
 *
 */
public String getProductName() {
   return productName;
}

/**
 * Sets the value of the productName property.
 *
 * @param value
 *     allowed object is
 *     {@link String }
 *
 */
public void setProductName(String value) {
   this.productName = value;
}

/**
 * Gets the value of the price property.
```

```
     *
     * @return
     *     possible object is
     *     {@link String }
     *
     */
    public String getPrice() {
        return price;
    }

    /**
     * Sets the value of the price property.
     *
     * @param value
     *     allowed object is
     *     {@link String }
     *
     */
    public void setPrice(String value) {
        this.price = value;
    }

}
```

/*

The following is the newOrderType XML value object, it is a XML format of the newOrder object

*/

NewOrderType Value Object:

```
package Order.Serializer;

import java.util.ArrayList;
import java.util.List;

import javax.xml.bind.annotation.XmlAccessType;
import javax.xml.bind.annotation.XmlAccessorType;
import javax.xml.bind.annotation.XmlElement;
import javax.xml.bind.annotation.XmlType;


/**
```

```
 * <p>Java class for NewOrderType complex type.
 *
 * <p>The following schema fragment specifies the expected content contained within
this class.
 *
 * <pre>
 * &lt;complexType name="NewOrderType">
 *   &lt;complexContent>
 *     &lt;restriction base="{http://www.w3.org/2001/XMLSchema}anyType">
 *       &lt;sequence>
 *         &lt;element name="IssueDate"
type="{http://www.w3.org/2001/XMLSchema}string"/>
 *         &lt;element name="CustomerParty" type="{}CustomerPartyType"/>
 *         &lt;element name="Total"
type="{http://www.w3.org/2001/XMLSchema}string"/>
 *         &lt;element name="Status"
type="{http://www.w3.org/2001/XMLSchema}string"/>
 *         &lt;element name="OrderLine" type="{}OrderLineType"
maxOccurs="unbounded"/>
 *       &lt;/sequence>
 *     &lt;/restriction>
 *   &lt;/complexContent>
 * &lt;/complexType>
 * </pre>
 *
 *
 */
@XmlAccessorType(XmlAccessType.FIELD)
@XmlType(name = "NewOrderType",
        propOrder = { "issueDate", "customerParty", "total", "status",
                    "orderLine" })
public class NewOrderType {

    @XmlElement(name = "IssueDate", required = true)
    protected String issueDate;
    @XmlElement(name = "CustomerParty", required = true)
    protected CustomerPartyType customerParty;
    @XmlElement(name = "Total", required = true)
    protected String total;
    @XmlElement(name = "Status", required = true)
    protected String status;
    @XmlElement(name = "OrderLine", required = true)
    protected List<OrderLineType> orderLine;

    /**
     * Gets the value of the issueDate property.
     *
     * @return
```

```
     *     possible object is
     *     {@link String }
     *
     */
    public String getIssueDate() {
        return issueDate;
    }

    /**
     * Sets the value of the issueDate property.
     *
     * @param value
     *     allowed object is
     *     {@link String }
     *
     */
    public void setIssueDate(String value) {
        this.issueDate = value;
    }

    /**
     * Gets the value of the customerParty property.
     *
     * @return
     *     possible object is
     *     {@link CustomerPartyType }
     *
     */
    public CustomerPartyType getCustomerParty() {
        return customerParty;
    }

    /**
     * Sets the value of the customerParty property.
     *
     * @param value
     *     allowed object is
     *     {@link CustomerPartyType }
     *
     */
    public void setCustomerParty(CustomerPartyType value) {
        this.customerParty = value;
    }

    /**
     * Gets the value of the total property.
     *
     * @return
```

```
     *     possible object is
     *     {@link String }
     *
     */
    public String getTotal() {
        return total;
    }

    /**
     * Sets the value of the total property.
     *
     * @param value
     *     allowed object is
     *     {@link String }
     *
     */
    public void setTotal(String value) {
        this.total = value;
    }

    /**
     * Gets the value of the status property.
     *
     * @return
     *     possible object is
     *     {@link String }
     *
     */
    public String getStatus() {
        return status;
    }

    /**
     * Sets the value of the status property.
     *
     * @param value
     *     allowed object is
     *     {@link String }
     *
     */
    public void setStatus(String value) {
        this.status = value;
    }
```

```
    /**
     * Gets the value of the orderLine property.
     *
     * <p>
     * This accessor method returns a reference to the live list,
     * not a snapshot. Therefore any modification you make to the
     * returned list will be present inside the JAXB object.
     * This is why there is not a <CODE>set</CODE> method for the orderLine
property.
     *
     * <p>
     * For example, to add a new item, do as follows:
     * <pre>
     *    getOrderLine().add(newItem);
     * </pre>
     *
     *
     * <p>
     * Objects of the following type(s) are allowed in the list
     * {@link OrderLineType }
     *
     *
     */
    public List<OrderLineType> getOrderLine() {
        if (orderLine == null) {
            orderLine = new ArrayList<OrderLineType>();
        }
        return this.orderLine;
    }

}


/*
The following is the XML object factory, it is use for create, modify the XML value
object
*/
```

Object Factory:

```
package Order.Serializer;

import javax.xml.bind.annotation.XmlRegistry;


/**
```

```
 * This object contains factory methods for each
 * Java content interface and Java element interface
 * generated in the orderxml package.
 * <p>An ObjectFactory allows you to programatically
 * construct new instances of the Java representation
 * for XML content. The Java representation of XML
 * content can consist of schema derived interfaces
 * and classes representing the binding of schema
 * type definitions, element declarations and model
 * groups.  Factory methods for each of these are
 * provided in this class.
 *
 */
@XmlRegistry
public class ObjectFactory {


    /**
     * Create a new ObjectFactory that can be used to create new instances of schema
derived classes for package: orderxml
     *
     */
    public ObjectFactory() {
    }

    /**
     * Create an instance of {@link ChangeOrderType }
     *
     */
    public ChangeOrderType createChangeOrderType() {
        return new ChangeOrderType();
    }

    /**
     * Create an instance of {@link CustomerPartyType }
     *
     */
    public CustomerPartyType createCustomerPartyType() {
        return new CustomerPartyType();
    }

    /**
     * Create an instance of {@link ItemType }
     *
     */
    public ItemType createItemType() {
        return new ItemType();
    }
```

```
    /**
     * Create an instance of {@link OrderLineType }
     *
     */
    public OrderLineType createOrderLineType() {
        return new OrderLineType();
    }

    /**
     * Create an instance of {@link Order }
     *
     */
    public OrderType createOrder() {
        return new OrderType();
    }

    /**
     * Create an instance of {@link ResponseOrderType }
     *
     */
    public ResponseOrderType createResponseOrderType() {
        return new ResponseOrderType();
    }

    /**
     * Create an instance of {@link NewOrderType }
     *
     */
    public NewOrderType createNewOrderType() {
        return new NewOrderType();
    }

}


/*
The following is the orderLineType XML value object, it is a XML format of the
orderlineType object
*/
```

OrderLineType Value Object:

```
package Order.Serializer;
```

```
import javax.xml.bind.annotation.XmlAccessType;
import javax.xml.bind.annotation.XmlAccessorType;
import javax.xml.bind.annotation.XmlAttribute;
import javax.xml.bind.annotation.XmlElement;
import javax.xml.bind.annotation.XmlType;


/**
 * <p>Java class for OrderLineType complex type.
 *
 * <p>The following schema fragment specifies the expected content contained within
this class.
 *
 * <pre>
 * &lt;complexType name="OrderLineType">
 *   &lt;complexContent>
 *     &lt;restriction base="{http://www.w3.org/2001/XMLSchema}anyType">
 *       &lt;sequence>
 *         &lt;element name="OrderLineNumber"
type="{http://www.w3.org/2001/XMLSchema}int"/>
 *         &lt;element name="Item" type="{}ItemType"/>
 *         &lt;element name="Quantity"
type="{http://www.w3.org/2001/XMLSchema}int"/>
 *       &lt;/sequence>
 *       &lt;attribute ref="{}Action use="required""/>
 *     &lt;/restriction>
 *   &lt;/complexContent>
 * &lt;/complexType>
 * </pre>
 *
 *
 */
@XmlAccessorType(XmlAccessType.FIELD)
@XmlType(name = "OrderLineType",
        propOrder = { "orderLineNumber", "item", "quantity" })
public class OrderLineType {
    @XmlElement(name = "OrderLineNumber")
    protected int orderLineNumber;
    @XmlElement(name = "Item", required = true)
    protected ItemType item;
    @XmlElement(name = "Quantity")
    protected int quantity;
    @XmlAttribute(name = "Action", required = true)
    protected String action;

    /**
     * Gets the value of the orderLineNumber property.
     *
```

```
 */
public int getOrderLineNumber() {
    return orderLineNumber;
}

/**
 * Sets the value of the orderLineNumber property.
 *
 */
public void setOrderLineNumber(int value) {
    this.orderLineNumber = value;
}

/**
 * Gets the value of the item property.
 *
 * @return
 *     possible object is
 *     {@link ItemType }
 *
 */
public ItemType getItem() {
    return item;
}

/**
 * Sets the value of the item property.
 *
 * @param value
 *     allowed object is
 *     {@link ItemType }
 *
 */
public void setItem(ItemType value) {
    this.item = value;
}

/**
 * Gets the value of the quantity property.
 *
 */
public int getQuantity() {
    return quantity;
}

/**
 * Sets the value of the quantity property.
 *
```

```
     */
    public void setQuantity(int value) {
        this.quantity = value;
    }

    /**
     * Gets the value of the action property.
     *
     * @return
     *     possible object is
     *     {@link String }
     *
     */
    public String getAction() {
        return action;
    }

    /**
     * Sets the value of the action property.
     *
     * @param value
     *     allowed object is
     *     {@link String }
     *
     */
    public void setAction(String value) {
        this.action = value;
    }

}


/*
The following is the orderType XML value object, it is a XML format of the order
object
*/
```

OrderType Value Object:

```
package Order.Serializer;

import javax.xml.bind.annotation.XmlAccessType;
import javax.xml.bind.annotation.XmlAccessorType;
import javax.xml.bind.annotation.XmlAttribute;
import javax.xml.bind.annotation.XmlElement;
```

```
import javax.xml.bind.annotation.XmlRootElement;
import javax.xml.bind.annotation.XmlType;


/**
 * <p>Java class for anonymous complex type.
 *
 * <p>The following schema fragment specifies the expected content contained within
this class.
 *
 * <pre>
 * &lt;complexType>
 *   &lt;complexContent>
 *     &lt;restriction base="{http://www.w3.org/2001/XMLSchema}anyType">
 *       &lt;choice>
 *         &lt;element name="NewOrder" type="{}NewOrderType" minOccurs="0"/>
 *         &lt;element name="ChangeOrder" type="{}ChangeOrderType"
minOccurs="0"/>
 *         &lt;element name="ResponseOrder" type="{}ResponseOrderType"
minOccurs="0"/>
 *       &lt;/choice>
 *       &lt;attribute name="ID" use="required"
type="{http://www.w3.org/2001/XMLSchema}int" />
 *       &lt;attribute ref="{}Action use="required""/>
 *     &lt;/restriction>
 *   &lt;/complexContent>
 * &lt;/complexType>
 * </pre>
 *
 *
 */
@XmlAccessorType(XmlAccessType.FIELD)
@XmlType(name = "", propOrder = { "newOrder", "changeOrder", "responseOrder" })
@XmlRootElement(name = "Order")
public class OrderType {

    @XmlElement(name = "NewOrder")
    protected NewOrderType newOrder;
    @XmlElement(name = "ChangeOrder")
    protected ChangeOrderType changeOrder;
    @XmlElement(name = "ResponseOrder")
    protected ResponseOrderType responseOrder;
    @XmlAttribute(name = "ID", required = true)
    protected int id;
    @XmlAttribute(name = "Action", required = true)
    protected String action;

    /**
```

```
     * Gets the value of the newOrder property.
     *
     * @return
     *     possible object is
     *     {@link NewOrderType }
     *
     */
    public NewOrderType getNewOrder() {
        return newOrder;
    }

    /**
     * Sets the value of the newOrder property.
     *
     * @param value
     *     allowed object is
     *     {@link NewOrderType }
     *
     */
    public void setNewOrder(NewOrderType value) {
        this.newOrder = value;
    }

    /**
     * Gets the value of the changeOrder property.
     *
     * @return
     *     possible object is
     *     {@link ChangeOrderType }
     *
     */
    public ChangeOrderType getChangeOrder() {
        return changeOrder;
    }

    /**
     * Sets the value of the changeOrder property.
     *
     * @param value
     *     allowed object is
     *     {@link ChangeOrderType }
     *
     */
    public void setChangeOrder(ChangeOrderType value) {
        this.changeOrder = value;
    }

    /**
```

```
     * Gets the value of the responseOrder property.
     *
     * @return
     *     possible object is
     *     {@link ResponseOrderType }
     *
     */
    public ResponseOrderType getResponseOrder() {
        return responseOrder;
    }

    /**
     * Sets the value of the responseOrder property.
     *
     * @param value
     *     allowed object is
     *     {@link ResponseOrderType }
     *
     */
    public void setResponseOrder(ResponseOrderType value) {
        this.responseOrder = value;
    }

    /**
     * Gets the value of the id property.
     *
     */
    public int getID() {
        return id;
    }

    /**
     * Sets the value of the id property.
     *
     */
    public void setID(int value) {
        this.id = value;
    }

    /**
     * Gets the value of the action property.
     *
     * @return
     *     possible object is
     *     {@link String }
     *
     */
    public String getAction() {
```

```
        return action;
    }

    /**
     * Sets the value of the action property.
     *
     * @param value
     *     allowed object is
     *     {@link String }
     *
     */
    public void setAction(String value) {
        this.action = value;
    }

}

/*
The following is the responseType XML value object, it is a XML format of the
response object
*/
```

ResponseType Value Object:

```
package Order.Serializer;

import javax.xml.bind.annotation.XmlAccessType;
import javax.xml.bind.annotation.XmlAccessorType;
import javax.xml.bind.annotation.XmlElement;
import javax.xml.bind.annotation.XmlType;
import javax.xml.bind.annotation.XmlRootElement;

/**
 * <p>Java class for ResponseOrderType complex type.
 *
 * <p>The following schema fragment specifies the expected content contained within
this class.
 *
 * <pre>
 * &lt;complexType name="ResponseOrderType">
 *   &lt;complexContent>
 *     &lt;restriction base="{http://www.w3.org/2001/XMLSchema}anyType">
 *       &lt;sequence>
 *         &lt;element name="AcceptedIndicator"
type="{http://www.w3.org/2001/XMLSchema}boolean" minOccurs="0"/>
 *       &lt;/sequence>
```

```
 *     &lt;/restriction>
 *   &lt;/complexContent>
 * &lt;/complexType>
 * </pre>
 *
 *
 */
@XmlAccessorType(XmlAccessType.FIELD)
@XmlType(name = "ResponseOrderType", propOrder = { "acceptedIndicator" })
@XmlRootElement(name = "ResponseOrder")
public class ResponseOrderType {

    @XmlElement(name = "AcceptedIndicator")
    protected Boolean acceptedIndicator;

    /**
     * Gets the value of the acceptedIndicator property.
     *
     * @return
     *     possible object is
     *     {@link Boolean }
     *
     */
    public Boolean isAcceptedIndicator() {
        return acceptedIndicator;
    }

    /**
     * Sets the value of the acceptedIndicator property.
     *
     * @param value
     *     allowed object is
     *     {@link Boolean }
     *
     */
    public void setAcceptedIndicator(Boolean value) {
        this.acceptedIndicator = value;
    }

}
```

Appendix 7: Purchase Order Schema of the Prototype application

```
/*
The following is the purchase order XML schema that define the business document
*/


<?xml version="1.0" encoding="iso-8859-1"?>
<!--Created with Liquid XML Studio - FREE Community Edition 7.0.5.906
(http://www.liquid-technologies.com)-->
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
 <xs:attribute name="Action">
  <xs:simpleType>
   <xs:restriction base="xs:string">
    <xs:enumeration value="new" />
    <xs:enumeration value="change" />
    <xs:enumeration value="cancel" />
    <xs:enumeration value="retrieve" />
   </xs:restriction>
  </xs:simpleType>
 </xs:attribute>
 <xs:complexType name="CustomerPartyType">
  <xs:sequence>
   <xs:element name="ID" type="xs:int" />
   <xs:element name="Name" type="xs:string" />
   <xs:element name="Address" type="xs:string" />
   <xs:element name="Contact" type="xs:string" />
  </xs:sequence>
 </xs:complexType>
 <xs:complexType name="ItemType">
  <xs:sequence>
   <xs:element name="ProductId" type="xs:int" />
   <xs:element name="ProductName" type="xs:string" />
   <xs:element name="Price" type="xs:string" />
  </xs:sequence>
 </xs:complexType>
 <xs:complexType name="NewOrderType">
  <xs:sequence>
   <xs:element name="IssueDate" type="xs:string" />
   <xs:element name="CustomerParty" type="CustomerPartyType" />
   <xs:element name="Total" type="xs:string" />
   <xs:element name="Status" type="xs:string" />
   <xs:element maxOccurs="unbounded" name="OrderLine" type="OrderLineType"
/>
  </xs:sequence>
 </xs:complexType>
 <xs:complexType name="ChangeOrderType">
  <xs:sequence>
```

```
    <xs:element minOccurs="1" maxOccurs="unbounded" name="OrderLine"
type="OrderLineType" />
   </xs:sequence>
  </xs:complexType>
  <xs:complexType name="OrderLineType">
   <xs:sequence>
    <xs:element name="OrderLineNumber" type="xs:int" />
    <xs:element name="Item" type="ItemType" />
    <xs:element name="Quantity" type="xs:int" />
   </xs:sequence>
   <xs:attribute ref="Action" use="required" />
  </xs:complexType>
  <xs:complexType name="ResponseOrderType">
   <xs:sequence>
    <xs:element minOccurs="0" name="AcceptedIndicator" type="xs:boolean" />
   </xs:sequence>
  </xs:complexType>
  <xs:element name="Order">
   <xs:complexType>
    <xs:choice>
     <xs:element minOccurs="0" name="NewOrder" type="NewOrderType" />
     <xs:element minOccurs="0" name="ChangeOrder" type="ChangeOrderType" />
     <xs:element minOccurs="0" name="ResponseOrder" type="ResponseOrderType"
/>
    </xs:choice>
    <xs:attribute name="ID" type="xs:int" use="required" />
    <xs:attribute ref="Action" use="required" />
   </xs:complexType>
  </xs:element>
</xs:schema>
```

Appendix 8: Object Serializer of the Prototype application

```
/*
The following is the serialiser class to convert the entity object to and from the XML
value object
*/

package Order.Serializer;

import Order.DAO.CustomerDAO;
import Order.DAO.OrderDAO;
import Order.DAO.ProductDAO;
import Order.Entity.Customer;
import Order.Entity.Order;
import Order.Entity.OrderLine;
import Order.Entity.Product;
import java.io.ByteArrayInputStream;
import java.io.ByteArrayOutputStream;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;
import java.util.List;
import java.util.logging.Level;
import java.util.logging.Logger;
import javax.persistence.EntityManager;
import javax.persistence.EntityManagerFactory;
import javax.persistence.Persistence;
import javax.swing.plaf.basic.BasicSliderUI.ActionScroller;
import javax.xml.bind.JAXBContext;
import javax.xml.bind.JAXBException;
import javax.xml.bind.Marshaller;
import javax.xml.bind.Unmarshaller;

public class OrderSerializer {

    private static final String PERSISTENCE_UNIT_NAME = "OrderEntityPU";
    private EntityManagerFactory factory;
    private EntityManager em;
    private CustomerDAO customerDAO;
    private OrderDAO orderDAO;
    private ProductDAO productDAO;
```

```
    /**
     * Constructor for the OrderSerializer for accessing the DAO.
     */
    public OrderSerializer() {
        factory =
Persistence.createEntityManagerFactory(PERSISTENCE_UNIT_NAME);
            em = factory.createEntityManager();
            customerDAO = new CustomerDAO(em);
            orderDAO = new OrderDAO(em);
            productDAO = new ProductDAO(em);
    }
    static JAXBContext context = null;

    /**
     * Export the Order object to XML document
     */
    public void exportToXML(Order order, String address) {
        File file = new File(address);
        OutputStream stream = null;
        try {
            stream = new FileOutputStream(file);
            this.createXMLDocument(order, stream);
        } catch (FileNotFoundException ex) {
            Logger.getLogger(OrderSerializer.class.getName()).log(Level.SEVERE, null,
ex);
        } finally {
            try {
                stream.close();
            } catch (IOException ex) {
                Logger.getLogger(OrderSerializer.class.getName()).log(Level.WARNING,
null, ex);
            }
        }
    }

    /**
     * Transfrom the Order object from entity to value object
     */
public void createXMLDocument(Order order, OutputStream stream) {
        try {
            context = JAXBContext.newInstance("Order.Serializer");
            Marshaller marshaller = context.createMarshaller();
            em.getTransaction().begin();

            OrderType purchaseOrder = new OrderType();
            purchaseOrder.setAction(order.getAction());
            purchaseOrder.setID(order.getId());
```

```
 //Create new order from action “new”
if (order.getAction().contains("new")) {
   NewOrderType newOrderType = new NewOrderType();
   newOrderType.setIssueDate(order.getCreateDate());
   newOrderType.setStatus(order.getStatus());
   newOrderType.setTotal(order.getTotal().toString());

   CustomerPartyType customerPartyType = new CustomerPartyType();

   Customer customer = order.getCustomer();
   if (customer != null) {
      customerPartyType.setID(customer.getId());
      customerPartyType.setName(customer.getName());
      customerPartyType.setAddress(customer.getAddress());
      customerPartyType.setContact(customer.getContact().toString());
   }
   newOrderType.setCustomerParty(customerPartyType);

   for (OrderLine orderline : (List<OrderLine>) order.getOrderlines()) {
      OrderLineType orderlineType = new OrderLineType();
      orderlineType.setOrderLineNumber(orderline.getId());
      orderlineType.setQuantity(orderline.getQuantity());

      ItemType itemType = new ItemType();

      Product product = new Product();
      product = productDAO.FindById(orderline.getProduct().getId());
      itemType.setProductId(product.getId());
      itemType.setProductName(product.getDescription());
      itemType.setPrice(product.getPrice().toString());
      orderlineType.setItem(itemType);

      newOrderType.getOrderLine().add(orderlineType);
   }
   purchaseOrder.setNewOrder(newOrderType);
}

//Create change order from action “change”
else if (order.getAction().contains("change")) {
   ChangeOrderType changeOrderType = new ChangeOrderType();
   for (OrderLine orderline : (List<OrderLine>) order.getOrderlines()) {
      OrderLineType orderlineType = new OrderLineType();
      orderlineType.setAction(orderline.getAction());
      orderlineType.setOrderLineNumber(orderline.getId());
      orderlineType.setQuantity(orderline.getQuantity());
      ItemType itemType = new ItemType();

      Product product = new Product();
```

```
                product = productDAO.FindById(orderline.getProduct().getId());
                itemType.setProductId(product.getId());
                itemType.setProductName(product.getDescription());
                itemType.setPrice(product.getPrice().toString());
                orderlineType.setItem(itemType);

                changeOrderType.getOrderLine().add(orderlineType);
            }
            purchaseOrder.setChangeOrder(changeOrderType);
        }
        marshaller.setProperty(Marshaller.JAXB_FORMATTED_OUTPUT,
Boolean.TRUE);
        marshaller.marshal(purchaseOrder, stream);

        // Commit the transaction, which will cause the entity to
        // be stored in the database
        em.getTransaction().commit();
      } catch (JAXBException e) {
        Logger.getLogger(OrderSerializer.class.getName()).log(Level.SEVERE, null,
e);
        System.out.println(e.toString());
        em.getTransaction().rollback();
      } catch (Exception e) {
        em.getTransaction().rollback();
      } finally {
        // It is always good practice to close the EntityManager so that
        // resources are conserved.
        em.close();
      }
    }

    /**
     * Process the incoming XML and return the response message
     */
    public String update(String request) throws Exception{
      ResponseOrderType response = new ResponseOrderType();

      InputStream stream = new ByteArrayInputStream(request.getBytes());
      OutputStream oStream = new ByteArrayOutputStream();

      //perform the logic
        Order order = this.importXML(stream);

        if(order != null)
        {
            response.setAcceptedIndicator(true);
        }
        else
```

```
            {
                response.setAcceptedIndicator(false);
            }

        //perform the XML display
        try {
            context = JAXBContext.newInstance("Order.Serializer");
            Marshaller marshaller = context.createMarshaller();
            marshaller.setProperty(Marshaller.JAXB_FORMATTED_OUTPUT,
Boolean.TRUE);
            marshaller.marshal(response, oStream);
        } catch (JAXBException ex) {
            Logger.getLogger(OrderSerializer.class.getName()).log(Level.SEVERE, null,
ex);
        }
        return oStream.toString();

    }

    /**
     * Process the incoming XML message and return the Order object
     */
    public Order importXML(String address) throws Exception {
        File xmlDocument = new File(address);
        Order order = null;
        FileInputStream fileInputStream = null;
        try {
            fileInputStream = new FileInputStream(xmlDocument);
            order = importXML(fileInputStream);
        } catch (FileNotFoundException ex) {
            Logger.getLogger(OrderSerializer.class.getName()).log(Level.SEVERE, null,
ex);
        } finally {
            try {
                if (fileInputStream != null) {
                    fileInputStream.close();
                }
            } catch (IOException ex) {
                Logger.getLogger(OrderSerializer.class.getName()).log(Level.SEVERE, null,
ex);
            }
        }
        return order;
    }

    /**
     * Process the XML message and create the order from the action
     */
```

```
   public Order importXML(InputStream stream) throws Exception{
        Order order = new Order();
        em.getTransaction().begin();

        context = JAXBContext.newInstance("Order.Serializer");

        Unmarshaller unmarshaller = context.createUnmarshaller();
        OrderType orderType = (OrderType) unmarshaller.unmarshal(stream);

        //create a new order from action “new”
        if(orderType.getAction().contains("new"))
        {
        NewOrderType newOrderType = orderType.getNewOrder();
        order.setAction(orderType.getAction());
        order.setId(orderType.getID());
        order.setCreateDate(newOrderType.getIssueDate());
        Customer customer =
customerDAO.FindById(newOrderType.getCustomerParty().getID());
        order.setCustomer(customer);
        order.setTotal(Integer.parseInt(newOrderType.getTotal()));
        order.setStatus(newOrderType.getStatus());

        //create new orderlines from the request message
        for (OrderLineType orderLineType : newOrderType.getOrderLine()) {
           OrderLine orderline = new OrderLine();
           orderline.setOrder(order);
           orderline.setId(orderLineType.getOrderLineNumber());
           orderline.setQuantity(orderLineType.getQuantity());
           Product product =
productDAO.FindById(orderLineType.getItem().getProductId());
           orderline.setProduct(product);
           order.addOrderline(orderline);
        }
        orderDAO.persist(order);
        }

        //change an order from action “change”
         else if (orderType.getAction().contains("change")) {
            ChangeOrderType changeOrderType = orderType.getChangeOrder();

        //update every orderlines from the request message
        for (OrderLineType orderLineType : changeOrderType.getOrderLine()) {
           OrderLine orderline = new OrderLine();
           orderline.setOrder(order);
           orderline.setId(orderLineType.getOrderLineNumber());
           orderline.setQuantity(orderLineType.getQuantity());
           Product product =
productDAO.FindById(orderLineType.getItem().getProductId());
```

```
            orderline.setProduct(product);

            order.addOrderline(orderline);
        }
            orderDAO.merge(order);
        }

        //cancel an order from action “cancel”
        else if (orderType.getAction().contains("cancel")) {
            order = orderDAO.FindById(orderType.getID());
            orderDAO.remove(order);
        }

        //retrieve an order from action “retrieve”
        else if (orderType.getAction().contains("retrieve")) {
            orderDAO.FindById(order.getId());
        }

        // Commit the transaction, which will cause the entity to
        // be stored in the database
        em.getTransaction().commit();
return order;
    }
}
```

Appendix 9: Prototype application Web Services

```
/*
The following is the Web Service class that define the processOrder WSDL interface
*/
package Order.Service;

import Order.Entity.Customer;
import Order.Serializer.OrderSerializer;
import javax.jws.WebService;
import javax.ejb.Stateless;
import javax.jws.WebMethod;
import javax.jws.WebParam;
import javax.jws.WebResult;

/**
 * This class expose the operation processOrder as a web service operation
 */
@WebService()
@Stateless()
public class OrderWebService {

    @WebMethod(operationName = "ProcessOrder")
    public String ProcessOrder(@WebParam(name = "request") String request) throws
Exception{
        String response = null;
            OrderSerializer serializer = new OrderSerializer();
            response = serializer.update(request);
        return response;
    }
}
```

Appendix 10: Prototype application Web Service Interface

```
/*
The following is process order WSDL interface document
*/

<?xml version="1.0" encoding="UTF-8"?>
<!-- Published by JAX-WS RI at http://jax-ws.dev.java.net. RI's version is JAX-WS RI
2.2-hudson-752-. -->
<!-- Generated by JAX-WS RI at http://jax-ws.dev.java.net. RI's version is JAX-WS RI
2.2-hudson-752-. -->
<definitions xmlns="http://schemas.xmlsoap.org/wsdl/"
xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/" xmlns:tns="http://Service.Order/"
xmlns:wsam="http://www.w3.org/2007/05/addressing/metadata"
xmlns:wsp="http://www.w3.org/ns/ws-policy"
xmlns:wsp1_2="http://schemas.xmlsoap.org/ws/2004/09/policy"
xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-
utility-1.0.xsd" xmlns:xsd="http://www.w3.org/2001/XMLSchema"
targetNamespace="http://Service.Order/" name="OrderWebServiceService">
  <wsp:Policy xmlns:wsat="http://schemas.xmlsoap.org/ws/2004/10/wsat"
wsu:Id="OrderWebServicePortBinding_findbyid_WSAT_Policy">
    <wsat:ATAlwaysCapability />
    <wsat:ATAssertion xmlns:ns1="http://schemas.xmlsoap.org/ws/2002/12/policy"
wsp:Optional="true" ns1:Optional="true" />
  </wsp:Policy>
  <types>
    <xsd:schema>
      <xsd:import namespace="http://Service.Order/"
schemaLocation="http://localhost:36772/OrderWebServiceService/OrderWebService?x
sd=1" />
    </xsd:schema>
  </types>
  <message name="findbyid">
    <part name="parameters" element="tns:findbyid" />
  </message>
  <message name="findbyidResponse">
    <part name="parameters" element="tns:findbyidResponse" />
  </message>
  <portType name="OrderWebService">
    <operation name="findbyid">
      <input wsam:Action="http://Service.Order/OrderWebService/findbyidRequest"
message="tns:findbyid" />
      <output wsam:Action="http://Service.Order/OrderWebService/findbyidResponse"
message="tns:findbyidResponse" />
    </operation>
  </portType>
  <binding name="OrderWebServicePortBinding" type="tns:OrderWebService">
```

```
    <soap:binding transport="http://schemas.xmlsoap.org/soap/http" style="document"
/>
    <operation name="findbyid">
      <wsp:PolicyReference
URI="#OrderWebServicePortBinding_findbyid_WSAT_Policy" />
      <soap:operation soapAction="" />
      <input>
        <wsp:PolicyReference
URI="#OrderWebServicePortBinding_findbyid_WSAT_Policy" />
        <soap:body use="literal" />
      </input>
      <output>
        <wsp:PolicyReference
URI="#OrderWebServicePortBinding_findbyid_WSAT_Policy" />
        <soap:body use="literal" />
      </output>
    </operation>
  </binding>
  <service name="OrderWebServiceService">
    <port name="OrderWebServicePort" binding="tns:OrderWebServicePortBinding">
      <soap:address
location="http://localhost:36772/OrderWebServiceService/OrderWebService" />
    </port>
  </service>
</definitions>
```

Appendix 11: Client's GUI of the Prototype application

```
/*
The following is the client use for invoking the process order Web Service, take the
XML request input and display the response output.
*/


package orderdesktop;

import Order.Entity.Customer;
import Order.Service.OrderWebService;
import java.awt.Color;
import java.io.StringWriter;
import org.jdesktop.application.Action;
import org.jdesktop.application.SingleFrameApplication;
import org.jdesktop.application.FrameView;
import javax.swing.JDialog;
import javax.swing.JFrame;
import javax.swing.UIManager;

/**
 * The application's main frame.
 */
public class OrderDesktopView extends FrameView {

    /**
     * Constructor for OrderDesktopView.
     */
    public OrderDesktopView(SingleFrameApplication app) {
        super(app);

        initComponents();
    }

    @Action
    public void showAboutBox() {
        if (aboutBox == null) {
            JFrame mainFrame = OrderDesktopApp.getApplication().getMainFrame();
            aboutBox = new OrderDesktopAboutBox(mainFrame);
            aboutBox.setLocationRelativeTo(mainFrame);
        }
        OrderDesktopApp.getApplication().show(aboutBox);
    }
```

```
/** This method is called from within the constructor to
 * initialize the form.
*/
@SuppressWarnings("unchecked")
private void initComponents() {

    mainPanel = new javax.swing.JPanel();
    jSplitPane1 = new javax.swing.JSplitPane();
    jSplitPane2 = new javax.swing.JSplitPane();
    jScrollPane1 = new javax.swing.JScrollPane();
    jTextArea1 = new javax.swing.JTextArea();
    jLabel1 = new javax.swing.JLabel();
    jSplitPane3 = new javax.swing.JSplitPane();
    jScrollPane2 = new javax.swing.JScrollPane();
    jTextArea2 = new javax.swing.JTextArea();
    jLabel2 = new javax.swing.JLabel();
    menuBar = new javax.swing.JMenuBar();
    javax.swing.JMenu fileMenu = new javax.swing.JMenu();
    javax.swing.JMenuItem exitMenuItem = new javax.swing.JMenuItem();
    javax.swing.JMenu helpMenu = new javax.swing.JMenu();
    javax.swing.JMenuItem aboutMenuItem = new javax.swing.JMenuItem();
    statusPanel = new javax.swing.JPanel();
    javax.swing.JSeparator statusPanelSeparator = new javax.swing.JSeparator();
    jButton1 = new javax.swing.JButton();
    jProgressBar1 = new javax.swing.JProgressBar();

    mainPanel.setName("mainPanel"); // NOI18N

    jSplitPane1.setDividerLocation(320);
    jSplitPane1.setContinuousLayout(true);
    jSplitPane1.setName("jSplitPane1"); // NOI18N

    jSplitPane2.setOrientation(javax.swing.JSplitPane.VERTICAL_SPLIT);
    jSplitPane2.setName("jSplitPane2"); // NOI18N

    jScrollPane1.setName("jScrollPane1"); // NOI18N

    jTextArea1.setColumns(20);
    jTextArea1.setLineWrap(true);
    jTextArea1.setRows(5);
    jTextArea1.setName("jTextArea1"); // NOI18N
    jScrollPane1.setViewportView(jTextArea1);

    jSplitPane2.setRightComponent(jScrollPane1);

    jLabel1.setHorizontalAlignment(javax.swing.SwingConstants.CENTER);
    jLabel1.setLabelFor(jScrollPane1);
```

```
        org.jdesktop.application.ResourceMap resourceMap =
org.jdesktop.application.Application.getInstance(orderdesktop.OrderDesktopApp.class).
getContext().getResourceMap(OrderDesktopView.class);
        jLabel1.setText(resourceMap.getString("jLabel1.text")); // NOI18N
        jLabel1.setName("jLabel1"); // NOI18N
        jSplitPane2.setLeftComponent(jLabel1);

        jSplitPane1.setTopComponent(jSplitPane2);

        jSplitPane3.setOrientation(javax.swing.JSplitPane.VERTICAL_SPLIT);
        jSplitPane3.setName("jSplitPane3"); // NOI18N

        jScrollPane2.setName("jScrollPane2"); // NOI18N

        jTextArea2.setColumns(20);
        jTextArea2.setLineWrap(true);
        jTextArea2.setRows(5);
        jTextArea2.setName("jTextArea2"); // NOI18N
        jScrollPane2.setViewportView(jTextArea2);

        jSplitPane3.setRightComponent(jScrollPane2);

        jLabel2.setHorizontalAlignment(javax.swing.SwingConstants.CENTER);
        jLabel2.setLabelFor(jScrollPane2);
        jLabel2.setText(resourceMap.getString("jLabel2.text")); // NOI18N
        jLabel2.setName("jLabel2"); // NOI18N
        jSplitPane3.setLeftComponent(jLabel2);

        jSplitPane1.setRightComponent(jSplitPane3);

        javax.swing.GroupLayout mainPanelLayout = new
javax.swing.GroupLayout(mainPanel);
        mainPanel.setLayout(mainPanelLayout);
        mainPanelLayout.setHorizontalGroup(

mainPanelLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING
)
            .addComponent(jSplitPane1, javax.swing.GroupLayout.DEFAULT_SIZE, 671,
Short.MAX_VALUE)
        );
        mainPanelLayout.setVerticalGroup(

mainPanelLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING
)
            .addComponent(jSplitPane1, javax.swing.GroupLayout.DEFAULT_SIZE, 446,
Short.MAX_VALUE)
        );
```

```
        menuBar.setName("menuBar"); // NOI18N

        fileMenu.setText(resourceMap.getString("fileMenu.text")); // NOI18N
        fileMenu.setName("fileMenu"); // NOI18N

        javax.swing.ActionMap actionMap =
org.jdesktop.application.Application.getInstance(orderdesktop.OrderDesktopApp.class).
getContext().getActionMap(OrderDesktopView.class, this);
        exitMenuItem.setAction(actionMap.get("quit")); // NOI18N
        exitMenuItem.setName("exitMenuItem"); // NOI18N
        fileMenu.add(exitMenuItem);

        menuBar.add(fileMenu);

        helpMenu.setText(resourceMap.getString("helpMenu.text")); // NOI18N
        helpMenu.setName("helpMenu"); // NOI18N

        aboutMenuItem.setAction(actionMap.get("showAboutBox")); // NOI18N
        aboutMenuItem.setName("aboutMenuItem"); // NOI18N
        helpMenu.add(aboutMenuItem);

        menuBar.add(helpMenu);

        statusPanel.setName("statusPanel"); // NOI18N

        statusPanelSeparator.setName("statusPanelSeparator"); // NOI18N

        jButton1.setText(resourceMap.getString("jButton1.text")); // NOI18N
        jButton1.setName("jButton1"); // NOI18N
        jButton1.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                jButton1ActionPerformed(evt);
            }
        });

        jProgressBar1.setName("jProgressBar1"); // NOI18N
        jProgressBar1.setStringPainted(true);

        javax.swing.GroupLayout statusPanelLayout = new
javax.swing.GroupLayout(statusPanel);
        statusPanel.setLayout(statusPanelLayout);
        statusPanelLayout.setHorizontalGroup(

statusPanelLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADIN
G)
            .addComponent(statusPanelSeparator,
javax.swing.GroupLayout.DEFAULT_SIZE, 671, Short.MAX_VALUE)
```

```
            .addGroup(javax.swing.GroupLayout.Alignment.TRAILING,
statusPanelLayout.createSequentialGroup()
                .addGap(43, 43, 43)
                .addComponent(jProgressBar1, javax.swing.GroupLayout.DEFAULT_SIZE,
384, Short.MAX_VALUE)
                .addGap(157, 157, 157)
                .addComponent(jButton1)
                .addGap(30, 30, 30))
        );
        statusPanelLayout.setVerticalGroup(

statusPanelLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADIN
G)
            .addGroup(javax.swing.GroupLayout.Alignment.TRAILING,
statusPanelLayout.createSequentialGroup()
                .addContainerGap()
                .addComponent(statusPanelSeparator,
javax.swing.GroupLayout.DEFAULT_SIZE, 2, Short.MAX_VALUE)

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.UNRELATED)

.addGroup(statusPanelLayout.createParallelGroup(javax.swing.GroupLayout.Alignmen
t.LEADING)
                    .addComponent(jButton1)
                    .addGroup(statusPanelLayout.createSequentialGroup()
                        .addComponent(jProgressBar1,
javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)
                        .addGap(6, 6, 6)))
                .addGap(11, 11, 11))
        );

        setComponent(mainPanel);
        setMenuBar(menuBar);
        setStatusBar(statusPanel);
    }// </editor-fold>


    /**
     * Button action for invoking the process order.
     */
    private void jButton1ActionPerformed(java.awt.event.ActionEvent evt) {

        String response = null;
        try{
            jProgressBar1.setValue(0);
        OrderWebService service = new OrderWebService();
        response = service.ProcessOrder(jTextArea1.getText());
```

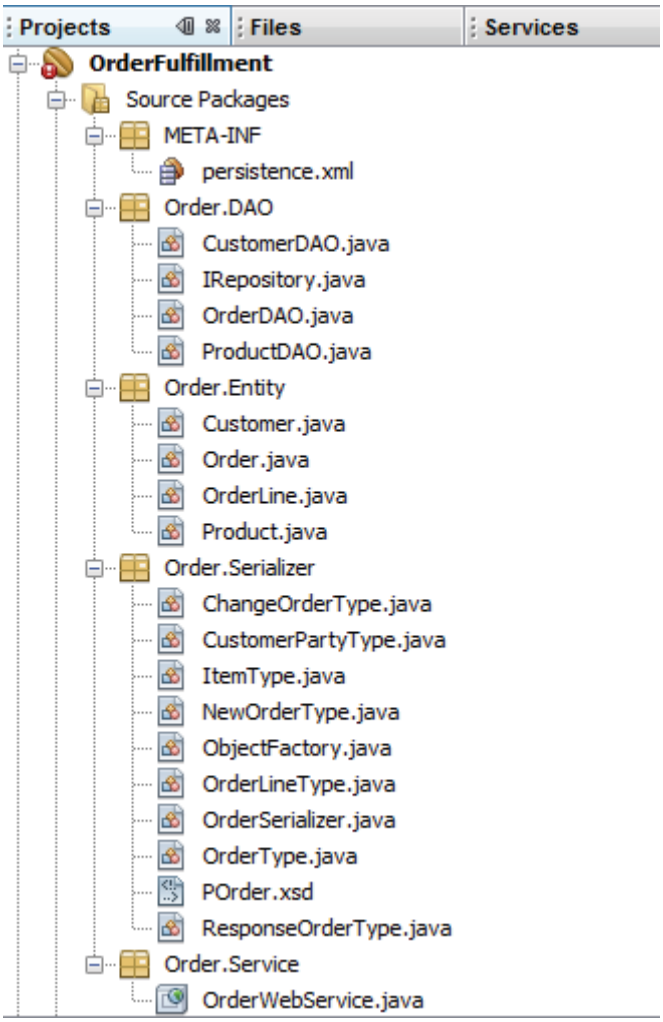
```
            jProgressBar1.setForeground(Color.blue);
            jProgressBar1.setValue(100);
            jProgressBar1.setString("Completed");
        }
        catch(Exception e){
            response = e.toString();
             jProgressBar1.setValue(100);
             jProgressBar1.setForeground(Color.red);
             jProgressBar1.setString("Failed");
        }
        jTextArea2.setText(response);
    }

    private javax.swing.JButton jButton1;
    private javax.swing.JLabel jLabel1;
    private javax.swing.JLabel jLabel2;
    private javax.swing.JProgressBar jProgressBar1;
    private javax.swing.JScrollPane jScrollPane1;
    private javax.swing.JScrollPane jScrollPane2;
    private javax.swing.JSplitPane jSplitPane1;
    private javax.swing.JSplitPane jSplitPane2;
    private javax.swing.JSplitPane jSplitPane3;
    private javax.swing.JTextArea jTextArea1;
    private javax.swing.JTextArea jTextArea2;
    private javax.swing.JPanel mainPanel;
    private javax.swing.JMenuBar menuBar;
    private javax.swing.JPanel statusPanel;

    private JDialog aboutBox;
}
```

Appendix 12: Overview of the Project Prototype application

/*

The following is project layout of the prototype in Netbean

*/

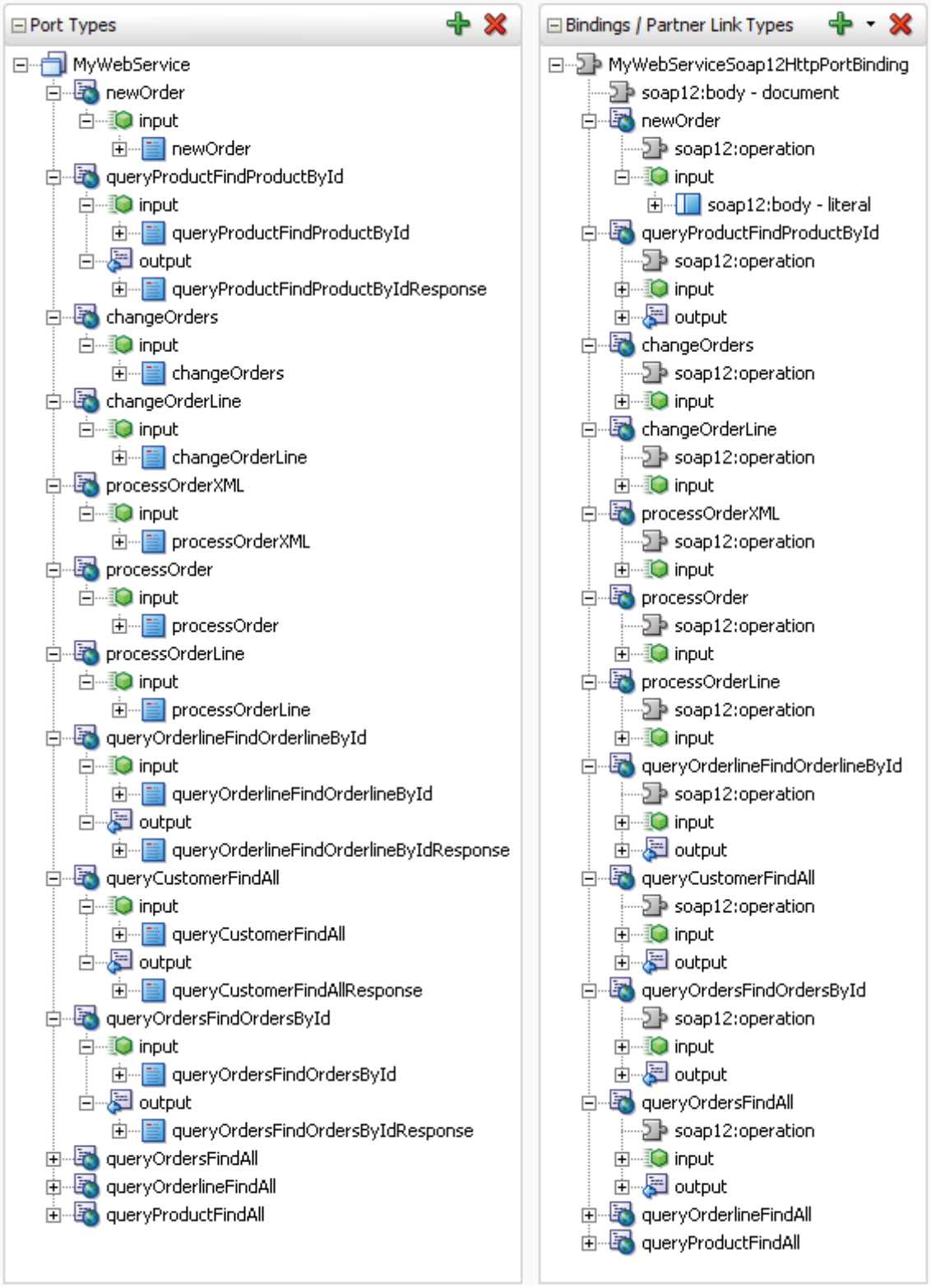


Appendix 13: Web Service interface showing available ports and operations

/*

The following is the available operations with their input and output

*/



Appendix 14: Sample UBL 2.0 Purchase Order example (Source: OAGIS)

```
<Order
xmlns:qdt="urn:oasis:names:specification:ubl:schema:xsd:QualifiedDatat
ypes-2"
xmlns:ccts="urn:oasis:names:specification:ubl:schema:xsd:CoreComponent
Parameters-
2"xmlns:cbc="urn:oasis:names:specification:ubl:schema:xsd:CommonBasicC
omponents-2"
xmlns:cac="urn:oasis:names:specification:ubl:schema:xsd:CommonAggregat
eComponents-
2"xmlns:udt="urn:un:unece:uncefact:data:draft:UnqualifiedDataTypesSche
maModule:2" xmlns="urn:oasis:names:specification:ubl:schema:xsd:Order-
2">
  <cbc:UBLVersionID>2.0</cbc:UBLVersionID>
  <cbc:CustomizationID>
urn:oasis:names:specification:ubl:xpath:Order-2.0:sbs-1.0-draft
</cbc:CustomizationID>
  <cbc:ProfileID>
bpid:urn:oasis:names:draft:bpss:ubl-2-sbs-order-with-simple-response-
draft
</cbc:ProfileID>
  <cbc:ID>AEG012345</cbc:ID>
  <cbc:SalesOrderID>CON0095678</cbc:SalesOrderID>
  <cbc:CopyIndicator>false</cbc:CopyIndicator>
  <cbc:UUID>6E09886B-DC6E-439F-82D1-7CCAC7F4E3B1</cbc:UUID>
  <cbc:IssueDate>2005-06-20</cbc:IssueDate>
  <cbc:Note>sample</cbc:Note>
  <cac:BuyerCustomerParty>

<cbc:CustomerAssignedAccountID>XFB01</cbc:CustomerAssignedAccountID>

<cbc:SupplierAssignedAccountID>GT00978567</cbc:SupplierAssignedAccount
ID>
    <cac:Party>
      <cac:PartyName>
        <cbc:Name>IYT Corporation</cbc:Name>
      </cac:PartyName>
      <cac:PostalAddress>
        <cbc:StreetName>Avon Way</cbc:StreetName>
        <cbc:BuildingName>Thereabouts</cbc:BuildingName>
        <cbc:BuildingNumber>56A</cbc:BuildingNumber>
        <cbc:CityName>Bridgtow</cbc:CityName>
        <cbc:PostalZone>ZZ99 1ZZ</cbc:PostalZone>
        <cbc:CountrySubentity>Avon</cbc:CountrySubentity>
        <cac:AddressLine>
          <cbc:Line>3rd Floor, Room 5</cbc:Line>
        </cac:AddressLine>
        <cac:Country>
          <cbc:IdentificationCode>GB</cbc:IdentificationCode>
        </cac:Country>
      </cac:PostalAddress>
      <cac:PartyTaxScheme>
        <cbc:RegistrationName>Bridgtow District
Council</cbc:RegistrationName>
        <cbc:CompanyID>12356478</cbc:CompanyID>
        <cbc:ExemptionReason>Local Authority</cbc:ExemptionReason>
        <cac:TaxScheme>
          <cbc:ID>UK VAT</cbc:ID>
```

```
          <cbc:TaxTypeCode>VAT</cbc:TaxTypeCode>
        </cac:TaxScheme>
      </cac:PartyTaxScheme>
      <cac:Contact>
        <cbc:Name>Mr Fred Churchill</cbc:Name>
        <cbc:Telephone>0127 2653214</cbc:Telephone>
        <cbc:Telefax>0127 2653215</cbc:Telefax>

<cbc:ElectronicMail>fred@iytcorporation.gov.uk</cbc:ElectronicMail>
      </cac:Contact>
    </cac:Party>
  </cac:BuyerCustomerParty>
  <cac:SellerSupplierParty>

<cbc:CustomerAssignedAccountID>CO001</cbc:CustomerAssignedAccountID>
    <cac:Party>
      <cac:PartyName>
        <cbc:Name>Consortial</cbc:Name>
      </cac:PartyName>
      <cac:PostalAddress>
        <cbc:StreetName>Busy Street</cbc:StreetName>
        <cbc:BuildingName>Thereabouts</cbc:BuildingName>
        <cbc:BuildingNumber>56A</cbc:BuildingNumber>
        <cbc:CityName>Farthing</cbc:CityName>
        <cbc:PostalZone>AA99 1BB</cbc:PostalZone>
        <cbc:CountrySubentity>Heremouthshire</cbc:CountrySubentity>
        <cac:AddressLine>
          <cbc:Line>The Roundabout</cbc:Line>
        </cac:AddressLine>
        <cac:Country>
          <cbc:IdentificationCode>GB</cbc:IdentificationCode>
        </cac:Country>
      </cac:PostalAddress>
      <cac:PartyTaxScheme>
        <cbc:RegistrationName>Farthing Purchasing
Consortia</cbc:RegistrationName>
        <cbc:CompanyID>175 269 2355</cbc:CompanyID>
        <cbc:ExemptionReason>N/A</cbc:ExemptionReason>
        <cac:TaxScheme>
          <cbc:ID>VAT</cbc:ID>
          <cbc:TaxTypeCode>VAT</cbc:TaxTypeCode>
        </cac:TaxScheme>
      </cac:PartyTaxScheme>
      <cac:Contact>
        <cbc:Name>Mrs Bouquet</cbc:Name>
        <cbc:Telephone>0158 1233714</cbc:Telephone>
        <cbc:Telefax>0158 1233856</cbc:Telefax>

<cbc:ElectronicMail>bouquet@fpconsortial.co.uk</cbc:ElectronicMail>
      </cac:Contact>
    </cac:Party>
  </cac:SellerSupplierParty>
  <cac:OriginatorCustomerParty>
    <cac:Party>
      <cac:PartyName>
        <cbc:Name>The Terminus</cbc:Name>
      </cac:PartyName>
      <cac:PostalAddress>
        <cbc:StreetName>Avon Way</cbc:StreetName>
        <cbc:BuildingName>Thereabouts</cbc:BuildingName>
        <cbc:BuildingNumber>56A</cbc:BuildingNumber>
```

```
        <cbc:CityName>Bridgtow</cbc:CityName>
        <cbc:PostalZone>ZZ99 1ZZ</cbc:PostalZone>
        <cbc:CountrySubentity>Avon</cbc:CountrySubentity>
        <cac:AddressLine>
          <cbc:Line>3rd Floor, Room 5</cbc:Line>
        </cac:AddressLine>
        <cac:Country>
          <cbc:IdentificationCode>GB</cbc:IdentificationCode>
        </cac:Country>
      </cac:PostalAddress>
      <cac:PartyTaxScheme>
        <cbc:RegistrationName>Bridgtow District
Council</cbc:RegistrationName>
        <cbc:CompanyID>12356478</cbc:CompanyID>
        <cbc:ExemptionReason>Local Authority</cbc:ExemptionReason>
        <cac:TaxScheme>
          <cbc:ID>UK VAT</cbc:ID>
          <cbc:TaxTypeCode>VAT</cbc:TaxTypeCode>
        </cac:TaxScheme>
      </cac:PartyTaxScheme>
      <cac:Contact>
        <cbc:Name>S Massiah</cbc:Name>
        <cbc:Telephone>0127 98876545</cbc:Telephone>
        <cbc:Telefax>0127 98876546</cbc:Telefax>
        <cbc:ElectronicMail>smassiah@the-
email.co.uk</cbc:ElectronicMail>
      </cac:Contact>
    </cac:Party>
  </cac:OriginatorCustomerParty>
  <cac:Delivery>
    <cac:DeliveryAddress>
      <cbc:StreetName>Avon Way</cbc:StreetName>
      <cbc:BuildingName>Thereabouts</cbc:BuildingName>
      <cbc:BuildingNumber>56A</cbc:BuildingNumber>
      <cbc:CityName>Bridgtow</cbc:CityName>
      <cbc:PostalZone>ZZ99 1ZZ</cbc:PostalZone>
      <cbc:CountrySubentity>Avon</cbc:CountrySubentity>
      <cac:AddressLine>
        <cbc:Line>3rd Floor, Room 5</cbc:Line>
      </cac:AddressLine>
      <cac:Country>
        <cbc:IdentificationCode>GB</cbc:IdentificationCode>
      </cac:Country>
    </cac:DeliveryAddress>
    <cac:RequestedDeliveryPeriod>
      <cbc:StartDate>2005-06-29</cbc:StartDate>
      <cbc:StartTime>09:30:47.0Z</cbc:StartTime>
      <cbc:EndDate>2005-06-29</cbc:EndDate>
      <cbc:EndTime>09:30:47.0Z</cbc:EndTime>
    </cac:RequestedDeliveryPeriod>
  </cac:Delivery>
  <cac:DeliveryTerms>
    <cbc:SpecialTerms>1% deduction for late delivery as per
contract</cbc:SpecialTerms>
  </cac:DeliveryTerms>
  <cac:TransactionConditions>
    <cbc:Description>
order response required; payment is by BACS or by cheque
</cbc:Description>
  </cac:TransactionConditions>
  <cac:AnticipatedMonetaryTotal>
```

```
    <cbc:LineExtensionAmount
currencyID="GBP">100.00</cbc:LineExtensionAmount>
    <cbc:PayableAmount currencyID="GBP">100.00</cbc:PayableAmount>
  </cac:AnticipatedMonetaryTotal>
  <cac:OrderLine>
    <cbc:Note>this is an illustrative order line</cbc:Note>
    <cac:LineItem>
      <cbc:ID>1</cbc:ID>
      <cbc:SalesOrderID>A</cbc:SalesOrderID>
      <cbc:LineStatusCode>NoStatus</cbc:LineStatusCode>
      <cbc:Quantity unitCode="KG">100</cbc:Quantity>
      <cbc:LineExtensionAmount
currencyID="GBP">100.00</cbc:LineExtensionAmount>
      <cbc:TotalTaxAmount currencyID="GBP">17.50</cbc:TotalTaxAmount>
      <cac:Price>
        <cbc:PriceAmount currencyID="GBP">100.00</cbc:PriceAmount>
        <cbc:BaseQuantity unitCode="KG">1</cbc:BaseQuantity>
      </cac:Price>
      <cac:Item>
        <cbc:Description>Acme beeswax</cbc:Description>
        <cbc:Name>beeswax</cbc:Name>
        <cac:BuyersItemIdentification>
          <cbc:ID>6578489</cbc:ID>
        </cac:BuyersItemIdentification>
        <cac:SellersItemIdentification>
          <cbc:ID>17589683</cbc:ID>
        </cac:SellersItemIdentification>
      </cac:Item>
    </cac:LineItem>
  </cac:OrderLine>
</Order>
```

Appendix 15: Sample UBL 2.0 Purchase Order Response example (Source: OAGIS)

```
<OrderResponseSimple
xmlns="urn:oasis:names:specification:ubl:schema:xsd:OrderResponseSimpl
e-
2"xmlns:cac="urn:oasis:names:specification:ubl:schema:xsd:CommonAggreg
ateComponents-2"
xmlns:cbc="urn:oasis:names:specification:ubl:schema:xsd:CommonBasicCom
ponents-
2"xmlns:ccts="urn:oasis:names:specification:ubl:schema:xsd:CoreCompone
ntParameters-2"
xmlns:qdt="urn:oasis:names:specification:ubl:schema:xsd:QualifiedDatat
ypes-
2"xmlns:udt="urn:un:unece:uncefact:data:draft:UnqualifiedDataTypesSche
maModule:2">
  <cbc:UBLVersionID>2.0</cbc:UBLVersionID>
  <cbc:CustomizationID>
urn:oasis:names:specification:ubl:xpath:OrderResponseSimple-2.0:sbs-
1.0-draft
</cbc:CustomizationID>
  <cbc:ProfileID>
bpid:urn:oasis:names:draft:bpss:ubl-2-sbs-order-with-simple-response-
draft
</cbc:ProfileID>
  <cbc:ID>66890-9-09</cbc:ID>
  <cbc:CopyIndicator>false</cbc:CopyIndicator>
  <cbc:UUID>569ED478-0EBE-4817-A234-DFB9ACA81218</cbc:UUID>
  <cbc:IssueDate>2005-06-20</cbc:IssueDate>
  <cbc:Note>sample</cbc:Note>
  <cbc:AcceptedIndicator>true</cbc:AcceptedIndicator>
  <cac:OrderReference>
    <cbc:ID>AEG012345</cbc:ID>
    <cbc:SalesOrderID>CON0095678</cbc:SalesOrderID>
    <cbc:UUID>6E09886B-DC6E-439F-82D1-7CCAC7F4E3B1</cbc:UUID>
    <cbc:IssueDate>2005-06-20</cbc:IssueDate>
  </cac:OrderReference>
  <cac:SellerSupplierParty>

<cbc:CustomerAssignedAccountID>CO001</cbc:CustomerAssignedAccountID>
    <cac:Party>
      <cac:PartyName>
        <cbc:Name>Consortial</cbc:Name>
      </cac:PartyName>
      <cac:PostalAddress>
        <cbc:StreetName>Busy Street</cbc:StreetName>
        <cbc:BuildingName>Thereabouts</cbc:BuildingName>
        <cbc:BuildingNumber>56A</cbc:BuildingNumber>
        <cbc:CityName>Farthing</cbc:CityName>
        <cbc:PostalZone>AA99 1BB</cbc:PostalZone>
        <cbc:CountrySubentity>Heremouthshire</cbc:CountrySubentity>
        <cac:AddressLine>
          <cbc:Line>The Roundabout</cbc:Line>
        </cac:AddressLine>
        <cac:Country>
          <cbc:IdentificationCode>GB</cbc:IdentificationCode>
        </cac:Country>
      </cac:PostalAddress>
      <cac:PartyTaxScheme>
```

```
        <cbc:RegistrationName>Farthing Purchasing
Consortia</cbc:RegistrationName>
        <cbc:CompanyID>175 269 2355</cbc:CompanyID>
        <cbc:ExemptionReason>N/A</cbc:ExemptionReason>
        <cac:TaxScheme>
          <cbc:ID>VAT</cbc:ID>
          <cbc:TaxTypeCode>VAT</cbc:TaxTypeCode>
        </cac:TaxScheme>
      </cac:PartyTaxScheme>
      <cac:Contact>
        <cbc:Name>Mrs Bouquet</cbc:Name>
        <cbc:Telephone>0158 1233714</cbc:Telephone>
        <cbc:Telefax>0158 1233856</cbc:Telefax>

<cbc:ElectronicMail>bouquet@fpconsortial.co.uk</cbc:ElectronicMail>
      </cac:Contact>
    </cac:Party>
  </cac:SellerSupplierParty>
  <cac:BuyerCustomerParty>

<cbc:CustomerAssignedAccountID>XFB01</cbc:CustomerAssignedAccountID>

<cbc:SupplierAssignedAccountID>GT00978567</cbc:SupplierAssignedAccount
ID>
    <cac:Party>
      <cac:PartyName>
        <cbc:Name>IYT Corporation</cbc:Name>
      </cac:PartyName>
      <cac:PostalAddress>
        <cbc:StreetName>Avon Way</cbc:StreetName>
        <cbc:BuildingName>Thereabouts</cbc:BuildingName>
        <cbc:BuildingNumber>56A</cbc:BuildingNumber>
        <cbc:CityName>Bridgtow</cbc:CityName>
        <cbc:PostalZone>ZZ99 1ZZ</cbc:PostalZone>
        <cbc:CountrySubentity>Avon</cbc:CountrySubentity>
        <cac:AddressLine>
          <cbc:Line>3rd Floor, Room 5</cbc:Line>
        </cac:AddressLine>
        <cac:Country>
          <cbc:IdentificationCode>GB</cbc:IdentificationCode>
        </cac:Country>
      </cac:PostalAddress>
      <cac:PartyTaxScheme>
        <cbc:RegistrationName>Bridgtow District
Council</cbc:RegistrationName>
        <cbc:CompanyID>12356478</cbc:CompanyID>
        <cbc:ExemptionReason>Local Authority</cbc:ExemptionReason>
        <cac:TaxScheme>
          <cbc:ID>UK VAT</cbc:ID>
          <cbc:TaxTypeCode>VAT</cbc:TaxTypeCode>
        </cac:TaxScheme>
      </cac:PartyTaxScheme>
      <cac:Contact>
        <cbc:Name>Mr Fred Churchill</cbc:Name>
        <cbc:Telephone>0127 2653214</cbc:Telephone>
        <cbc:Telefax>0127 2653215</cbc:Telefax>

<cbc:ElectronicMail>fred@iytcorporation.gov.uk</cbc:ElectronicMail>
      </cac:Contact>
    </cac:Party>
  </cac:BuyerCustomerParty>
```

```
  <cac:OriginatorCustomerParty>
    <cac:Party>
      <cac:PartyName>
        <cbc:Name>The Terminus</cbc:Name>
      </cac:PartyName>
      <cac:PostalAddress>
        <cbc:StreetName>Avon Way</cbc:StreetName>
        <cbc:BuildingName>Thereabouts</cbc:BuildingName>
        <cbc:BuildingNumber>56A</cbc:BuildingNumber>
        <cbc:CityName>Bridgtow</cbc:CityName>
        <cbc:PostalZone>ZZ99 1ZZ</cbc:PostalZone>
        <cbc:CountrySubentity>Avon</cbc:CountrySubentity>
        <cac:AddressLine>
          <cbc:Line>3rd Floor, Room 5</cbc:Line>
        </cac:AddressLine>
        <cac:Country>
          <cbc:IdentificationCode>GB</cbc:IdentificationCode>
        </cac:Country>
      </cac:PostalAddress>
      <cac:PartyTaxScheme>
        <cbc:RegistrationName>Bridgtow District
Council</cbc:RegistrationName>
        <cbc:CompanyID>12356478</cbc:CompanyID>
        <cbc:ExemptionReason>Local Authority</cbc:ExemptionReason>
        <cac:TaxScheme>
          <cbc:ID>UK VAT</cbc:ID>
          <cbc:TaxTypeCode>VAT</cbc:TaxTypeCode>
        </cac:TaxScheme>
      </cac:PartyTaxScheme>
      <cac:Contact>
        <cbc:Name>S Massiah</cbc:Name>
        <cbc:Telephone>0127 98876545</cbc:Telephone>
        <cbc:Telefax>0127 98876546</cbc:Telefax>
        <cbc:ElectronicMail>smassiah@the-
email.co.uk</cbc:ElectronicMail>
      </cac:Contact>
    </cac:Party>
  </cac:OriginatorCustomerParty>
</OrderResponseSimple>
```

Appendix 16: Sample UBL 2.0 Change Purchase Order Response example (Source: OAGIS)

```
<OrderChange>
  <ext:UBLExtensions>
    <ext:UBLExtension>
      <cbc:ID>normalizedString</cbc:ID>
      <cbc:Name>string</cbc:Name>

<ext:ExtensionAgencyID>normalizedString</ext:ExtensionAgencyID>
      <ext:ExtensionAgencyName>string</ext:ExtensionAgencyName>

<ext:ExtensionVersionID>normalizedString</ext:ExtensionVersionID>

<ext:ExtensionAgencyURI>normalizedString</ext:ExtensionAgencyURI>
      <ext:ExtensionURI>normalizedString</ext:ExtensionURI>

<ext:ExtensionReasonCode>normalizedString</ext:ExtensionReasonCode>
      <ext:ExtensionReason>string</ext:ExtensionReason>
      <ext:ExtensionContent>
        <!--any element-->
      </ext:ExtensionContent>
    </ext:UBLExtension>
  </ext:UBLExtensions>
  <cbc:UBLVersionID>normalizedString</cbc:UBLVersionID>
  <cbc:CustomizationID>normalizedString</cbc:CustomizationID>
  <cbc:ProfileID>normalizedString</cbc:ProfileID>
  <cbc:ID>normalizedString</cbc:ID>
  <cbc:SalesOrderID>normalizedString</cbc:SalesOrderID>
  <cbc:CopyIndicator>true</cbc:CopyIndicator>
  <cbc:UUID>normalizedString</cbc:UUID>
  <cbc:IssueDate>2000-01-01</cbc:IssueDate>
  <cbc:IssueTime>12:00:00</cbc:IssueTime>
  <cbc:SequenceNumberID>normalizedString</cbc:SequenceNumberID>
  <cbc:Note>string</cbc:Note>

<cbc:RequestedInvoiceCurrencyCode>normalizedString</cbc:RequestedIn
voiceCurrencyCode>

<cbc:DocumentCurrencyCode>normalizedString</cbc:DocumentCurrencyCod
e>

<cbc:PricingCurrencyCode>normalizedString</cbc:PricingCurrencyCode>
  <cbc:TaxCurrencyCode>normalizedString</cbc:TaxCurrencyCode>
  <cbc:CustomerReference>string</cbc:CustomerReference>
  <cbc:AccountingCostCode>normalizedString</cbc:AccountingCostCode>
  <cbc:AccountingCost>string</cbc:AccountingCost>
  <cbc:LineCountNumeric>1.0</cbc:LineCountNumeric>
  <cac:ValidityPeriod>
    <cbc:StartDate>2000-01-01</cbc:StartDate>
    <cbc:StartTime>12:00:00</cbc:StartTime>
    <cbc:EndDate>2000-01-01</cbc:EndDate>
    <cbc:EndTime>12:00:00</cbc:EndTime>
    <cbc:DurationMeasure unitCode="04">1.0</cbc:DurationMeasure>
    <cbc:DescriptionCode>normalizedString</cbc:DescriptionCode>
    <cbc:Description>string</cbc:Description>
```

```
  </cac:ValidityPeriod>
  <cac:OrderReference>
    <cbc:ID>normalizedString</cbc:ID>
    <cbc:SalesOrderID>normalizedString</cbc:SalesOrderID>
    <cbc:CopyIndicator>true</cbc:CopyIndicator>
    <cbc:UUID>normalizedString</cbc:UUID>
    <cbc:IssueDate>2000-01-01</cbc:IssueDate>
    <cbc:IssueTime>12:00:00</cbc:IssueTime>
    <cbc:CustomerReference>string</cbc:CustomerReference>
    <cac:DocumentReference>
      <cbc:ID>normalizedString</cbc:ID>
      <cbc:CopyIndicator>true</cbc:CopyIndicator>
      <cbc:UUID>normalizedString</cbc:UUID>
      <cbc:IssueDate>2000-01-01</cbc:IssueDate>
      <cbc:DocumentTypeCode>normalizedString</cbc:DocumentTypeCode>
      <cbc:DocumentType>string</cbc:DocumentType>
      <cbc:XPath>string</cbc:XPath>
      <cac:Attachment>...          </cac:Attachment>
    </cac:DocumentReference>
  </cac:OrderReference>
  <cac:QuotationDocumentReference>
    <cbc:ID>normalizedString</cbc:ID>
    <cbc:CopyIndicator>true</cbc:CopyIndicator>
    <cbc:UUID>normalizedString</cbc:UUID>
    <cbc:IssueDate>2000-01-01</cbc:IssueDate>
    <cbc:DocumentTypeCode>normalizedString</cbc:DocumentTypeCode>
    <cbc:DocumentType>string</cbc:DocumentType>
    <cbc:XPath>string</cbc:XPath>
    <cac:Attachment>
      <cbc:EmbeddedDocumentBinaryObject
mimeCode="application/CSTAdata+xml">GpM7</cbc:EmbeddedDocumentBinar
yObject>
      <cac:ExternalReference>...          </cac:ExternalReference>
    </cac:Attachment>
  </cac:QuotationDocumentReference>
  <cac:OriginatorDocumentReference>
    <cbc:ID>normalizedString</cbc:ID>
    <cbc:CopyIndicator>true</cbc:CopyIndicator>
    <cbc:UUID>normalizedString</cbc:UUID>
    <cbc:IssueDate>2000-01-01</cbc:IssueDate>
    <cbc:DocumentTypeCode>normalizedString</cbc:DocumentTypeCode>
    <cbc:DocumentType>string</cbc:DocumentType>
    <cbc:XPath>string</cbc:XPath>
    <cac:Attachment>
      <cbc:EmbeddedDocumentBinaryObject
mimeCode="application/CSTAdata+xml">GpM7</cbc:EmbeddedDocumentBinar
yObject>
      <cac:ExternalReference>...          </cac:ExternalReference>
    </cac:Attachment>
  </cac:OriginatorDocumentReference>
  <cac:AdditionalDocumentReference>
    <cbc:ID>normalizedString</cbc:ID>
    <cbc:CopyIndicator>true</cbc:CopyIndicator>
    <cbc:UUID>normalizedString</cbc:UUID>
    <cbc:IssueDate>2000-01-01</cbc:IssueDate>
    <cbc:DocumentTypeCode>normalizedString</cbc:DocumentTypeCode>
    <cbc:DocumentType>string</cbc:DocumentType>
    <cbc:XPath>string</cbc:XPath>
```

```
    <cac:Attachment>
      <cbc:EmbeddedDocumentBinaryObject
mimeCode="application/CSTAdata+xml">GpM7</cbc:EmbeddedDocumentBinar
yObject>
      <cac:ExternalReference>...          </cac:ExternalReference>
    </cac:Attachment>
  </cac:AdditionalDocumentReference>
  <cac:Contract>
    <cbc:ID>normalizedString</cbc:ID>
    <cbc:IssueDate>2000-01-01</cbc:IssueDate>
    <cbc:IssueTime>12:00:00</cbc:IssueTime>
    <cbc:ContractTypeCode>normalizedString</cbc:ContractTypeCode>
    <cbc:ContractType>string</cbc:ContractType>
    <cac:ValidityPeriod>
      <cbc:StartDate>2000-01-01</cbc:StartDate>
      <cbc:StartTime>12:00:00</cbc:StartTime>
      <cbc:EndDate>2000-01-01</cbc:EndDate>
      <cbc:EndTime>12:00:00</cbc:EndTime>
      <cbc:DurationMeasure unitCode="04">1.0</cbc:DurationMeasure>
      <cbc:DescriptionCode>normalizedString</cbc:DescriptionCode>
      <cbc:Description>string</cbc:Description>
    </cac:ValidityPeriod>
    <cac:ContractDocumentReference>
      <cbc:ID>normalizedString</cbc:ID>
      <cbc:CopyIndicator>true</cbc:CopyIndicator>
      <cbc:UUID>normalizedString</cbc:UUID>
      <cbc:IssueDate>2000-01-01</cbc:IssueDate>
      <cbc:DocumentTypeCode>normalizedString</cbc:DocumentTypeCode>
      <cbc:DocumentType>string</cbc:DocumentType>
      <cbc:XPath>string</cbc:XPath>
      <cac:Attachment>...          </cac:Attachment>
    </cac:ContractDocumentReference>
  </cac:Contract>
  <cac:Signature>
    <cbc:ID>normalizedString</cbc:ID>
    <cbc:Note>string</cbc:Note>
    <cbc:ValidationDate>2000-01-01</cbc:ValidationDate>
    <cbc:ValidationTime>12:00:00</cbc:ValidationTime>
    <cbc:ValidatorID>normalizedString</cbc:ValidatorID>
    <cbc:CanonicalizationMethod>string</cbc:CanonicalizationMethod>
    <cbc:SignatureMethod>string</cbc:SignatureMethod>
    <cac:SignatoryParty>
      <cbc:MarkCareIndicator>true</cbc:MarkCareIndicator>
      <cbc:MarkAttentionIndicator>true</cbc:MarkAttentionIndicator>
      <cbc:WebsiteURI>normalizedString</cbc:WebsiteURI>
      <cbc:LogoReferenceID>normalizedString</cbc:LogoReferenceID>
      <cbc:EndpointID>normalizedString</cbc:EndpointID>
      <cac:PartyIdentification>...
</cac:PartyIdentification>
      <cac:PartyName>...          </cac:PartyName>
      <cac:Language>...          </cac:Language>
      <cac:PostalAddress>...          </cac:PostalAddress>
      <cac:PhysicalLocation>...          </cac:PhysicalLocation>
      <cac:PartyTaxScheme>...          </cac:PartyTaxScheme>
      <cac:PartyLegalEntity>...          </cac:PartyLegalEntity>
      <cac:Contact>...          </cac:Contact>
      <cac:Person>...          </cac:Person>
      <cac:AgentParty>...          </cac:AgentParty>
```

```
    </cac:SignatoryParty>
    <cac:DigitalSignatureAttachment>
      <cbc:EmbeddedDocumentBinaryObject
mimeCode="application/CSTAdata+xml">GpM7</cbc:EmbeddedDocumentBinar
yObject>
      <cac:ExternalReference>...          </cac:ExternalReference>
    </cac:DigitalSignatureAttachment>
    <cac:OriginalDocumentReference>
      <cbc:ID>normalizedString</cbc:ID>
      <cbc:CopyIndicator>true</cbc:CopyIndicator>
      <cbc:UUID>normalizedString</cbc:UUID>
      <cbc:IssueDate>2000-01-01</cbc:IssueDate>
      <cbc:DocumentTypeCode>normalizedString</cbc:DocumentTypeCode>
      <cbc:DocumentType>string</cbc:DocumentType>
      <cbc:XPath>string</cbc:XPath>
      <cac:Attachment>...          </cac:Attachment>
    </cac:OriginalDocumentReference>
  </cac:Signature>
  <cac:BuyerCustomerParty>

<cbc:CustomerAssignedAccountID>normalizedString</cbc:CustomerAssign
edAccountID>

<cbc:SupplierAssignedAccountID>normalizedString</cbc:SupplierAssign
edAccountID>

<cbc:AdditionalAccountID>normalizedString</cbc:AdditionalAccountID>
    <cac:Party>
      <cbc:MarkCareIndicator>true</cbc:MarkCareIndicator>
      <cbc:MarkAttentionIndicator>true</cbc:MarkAttentionIndicator>
      <cbc:WebsiteURI>normalizedString</cbc:WebsiteURI>
      <cbc:LogoReferenceID>normalizedString</cbc:LogoReferenceID>
      <cbc:EndpointID>normalizedString</cbc:EndpointID>
      <cac:PartyIdentification>...
</cac:PartyIdentification>
      <cac:PartyName>...          </cac:PartyName>
      <cac:Language>...          </cac:Language>
      <cac:PostalAddress>...          </cac:PostalAddress>
      <cac:PhysicalLocation>...          </cac:PhysicalLocation>
      <cac:PartyTaxScheme>...          </cac:PartyTaxScheme>
      <cac:PartyLegalEntity>...          </cac:PartyLegalEntity>
      <cac:Contact>...          </cac:Contact>
      <cac:Person>...          </cac:Person>
      <cac:AgentParty>...          </cac:AgentParty>
    </cac:Party>
    <cac:DeliveryContact>
      <cbc:ID>normalizedString</cbc:ID>
      <cbc:Name>string</cbc:Name>
      <cbc:Telephone>string</cbc:Telephone>
      <cbc:Telefax>string</cbc:Telefax>
      <cbc:ElectronicMail>string</cbc:ElectronicMail>
      <cbc:Note>string</cbc:Note>
      <cac:OtherCommunication>...
</cac:OtherCommunication>
    </cac:DeliveryContact>
    <cac:AccountingContact>
      <cbc:ID>normalizedString</cbc:ID>
      <cbc:Name>string</cbc:Name>
```

```
      <cbc:Telephone>string</cbc:Telephone>
      <cbc:Telefax>string</cbc:Telefax>
      <cbc:ElectronicMail>string</cbc:ElectronicMail>
      <cbc:Note>string</cbc:Note>
      <cac:OtherCommunication>...
</cac:OtherCommunication>
    </cac:AccountingContact>
    <cac:BuyerContact>
      <cbc:ID>normalizedString</cbc:ID>
      <cbc:Name>string</cbc:Name>
      <cbc:Telephone>string</cbc:Telephone>
      <cbc:Telefax>string</cbc:Telefax>
      <cbc:ElectronicMail>string</cbc:ElectronicMail>
      <cbc:Note>string</cbc:Note>
      <cac:OtherCommunication>...
</cac:OtherCommunication>
    </cac:BuyerContact>
  </cac:BuyerCustomerParty>
  <cac:SellerSupplierParty>

<cbc:CustomerAssignedAccountID>normalizedString</cbc:CustomerAssign
edAccountID>

<cbc:AdditionalAccountID>normalizedString</cbc:AdditionalAccountID>
    <cbc:DataSendingCapability>string</cbc:DataSendingCapability>
    <cac:Party>
      <cbc:MarkCareIndicator>true</cbc:MarkCareIndicator>
      <cbc:MarkAttentionIndicator>true</cbc:MarkAttentionIndicator>
      <cbc:WebsiteURI>normalizedString</cbc:WebsiteURI>
      <cbc:LogoReferenceID>normalizedString</cbc:LogoReferenceID>
      <cbc:EndpointID>normalizedString</cbc:EndpointID>
      <cac:PartyIdentification>...
</cac:PartyIdentification>
      <cac:PartyName>...          </cac:PartyName>
      <cac:Language>...          </cac:Language>
      <cac:PostalAddress>...          </cac:PostalAddress>
      <cac:PhysicalLocation>...          </cac:PhysicalLocation>
      <cac:PartyTaxScheme>...          </cac:PartyTaxScheme>
      <cac:PartyLegalEntity>...          </cac:PartyLegalEntity>
      <cac:Contact>...          </cac:Contact>
      <cac:Person>...          </cac:Person>
      <cac:AgentParty>...          </cac:AgentParty>
    </cac:Party>
    <cac:DespatchContact>
      <cbc:ID>normalizedString</cbc:ID>
      <cbc:Name>string</cbc:Name>
      <cbc:Telephone>string</cbc:Telephone>
      <cbc:Telefax>string</cbc:Telefax>
      <cbc:ElectronicMail>string</cbc:ElectronicMail>
      <cbc:Note>string</cbc:Note>
      <cac:OtherCommunication>...
</cac:OtherCommunication>
    </cac:DespatchContact>
    <cac:AccountingContact>
      <cbc:ID>normalizedString</cbc:ID>
      <cbc:Name>string</cbc:Name>
      <cbc:Telephone>string</cbc:Telephone>
      <cbc:Telefax>string</cbc:Telefax>
```

```
      <cbc:ElectronicMail>string</cbc:ElectronicMail>
      <cbc:Note>string</cbc:Note>
      <cac:OtherCommunication>...
</cac:OtherCommunication>
    </cac:AccountingContact>
    <cac:SellerContact>
      <cbc:ID>normalizedString</cbc:ID>
      <cbc:Name>string</cbc:Name>
      <cbc:Telephone>string</cbc:Telephone>
      <cbc:Telefax>string</cbc:Telefax>
      <cbc:ElectronicMail>string</cbc:ElectronicMail>
      <cbc:Note>string</cbc:Note>
      <cac:OtherCommunication>...
</cac:OtherCommunication>
    </cac:SellerContact>
  </cac:SellerSupplierParty>
  <cac:OriginatorCustomerParty>

<cbc:CustomerAssignedAccountID>normalizedString</cbc:CustomerAssign
edAccountID>

<cbc:SupplierAssignedAccountID>normalizedString</cbc:SupplierAssign
edAccountID>

<cbc:AdditionalAccountID>normalizedString</cbc:AdditionalAccountID>
    <cac:Party>
      <cbc:MarkCareIndicator>true</cbc:MarkCareIndicator>
      <cbc:MarkAttentionIndicator>true</cbc:MarkAttentionIndicator>
      <cbc:WebsiteURI>normalizedString</cbc:WebsiteURI>
      <cbc:LogoReferenceID>normalizedString</cbc:LogoReferenceID>
      <cbc:EndpointID>normalizedString</cbc:EndpointID>
      <cac:PartyIdentification>...
</cac:PartyIdentification>
      <cac:PartyName>...          </cac:PartyName>
      <cac:Language>...          </cac:Language>
      <cac:PostalAddress>...          </cac:PostalAddress>
      <cac:PhysicalLocation>...          </cac:PhysicalLocation>
      <cac:PartyTaxScheme>...          </cac:PartyTaxScheme>
      <cac:PartyLegalEntity>...          </cac:PartyLegalEntity>
      <cac:Contact>...          </cac:Contact>
      <cac:Person>...          </cac:Person>
      <cac:AgentParty>...          </cac:AgentParty>
    </cac:Party>
    <cac:DeliveryContact>
      <cbc:ID>normalizedString</cbc:ID>
      <cbc:Name>string</cbc:Name>
      <cbc:Telephone>string</cbc:Telephone>
      <cbc:Telefax>string</cbc:Telefax>
      <cbc:ElectronicMail>string</cbc:ElectronicMail>
      <cbc:Note>string</cbc:Note>
      <cac:OtherCommunication>...
</cac:OtherCommunication>
    </cac:DeliveryContact>
    <cac:AccountingContact>
      <cbc:ID>normalizedString</cbc:ID>
      <cbc:Name>string</cbc:Name>
      <cbc:Telephone>string</cbc:Telephone>
      <cbc:Telefax>string</cbc:Telefax>
```

```
      <cbc:ElectronicMail>string</cbc:ElectronicMail>
      <cbc:Note>string</cbc:Note>
      <cac:OtherCommunication>...
</cac:OtherCommunication>
    </cac:AccountingContact>
    <cac:BuyerContact>
      <cbc:ID>normalizedString</cbc:ID>
      <cbc:Name>string</cbc:Name>
      <cbc:Telephone>string</cbc:Telephone>
      <cbc:Telefax>string</cbc:Telefax>
      <cbc:ElectronicMail>string</cbc:ElectronicMail>
      <cbc:Note>string</cbc:Note>
      <cac:OtherCommunication>...
</cac:OtherCommunication>
    </cac:BuyerContact>
  </cac:OriginatorCustomerParty>
  <cac:FreightForwarderParty>
    <cbc:MarkCareIndicator>true</cbc:MarkCareIndicator>
    <cbc:MarkAttentionIndicator>true</cbc:MarkAttentionIndicator>
    <cbc:WebsiteURI>normalizedString</cbc:WebsiteURI>
    <cbc:LogoReferenceID>normalizedString</cbc:LogoReferenceID>
    <cbc:EndpointID>normalizedString</cbc:EndpointID>
    <cac:PartyIdentification>
      <cbc:ID>normalizedString</cbc:ID>
    </cac:PartyIdentification>
    <cac:PartyName>
      <cbc:Name>string</cbc:Name>
    </cac:PartyName>
    <cac:Language>
      <cbc:ID>normalizedString</cbc:ID>
      <cbc:Name>string</cbc:Name>
      <cbc:LocaleCode>normalizedString</cbc:LocaleCode>
    </cac:Language>
    <cac:PostalAddress>
      <cbc:ID>normalizedString</cbc:ID>
      <cbc:AddressTypeCode>normalizedString</cbc:AddressTypeCode>

<cbc:AddressFormatCode>normalizedString</cbc:AddressFormatCode>
      <cbc:Postbox>string</cbc:Postbox>
      <cbc:Floor>string</cbc:Floor>
      <cbc:Room>string</cbc:Room>
      <cbc:StreetName>string</cbc:StreetName>
      <cbc:AdditionalStreetName>string</cbc:AdditionalStreetName>
      <cbc:BlockName>string</cbc:BlockName>
      <cbc:BuildingName>string</cbc:BuildingName>
      <cbc:BuildingNumber>string</cbc:BuildingNumber>
      <cbc:InhouseMail>string</cbc:InhouseMail>
      <cbc:Department>string</cbc:Department>
      <cbc:MarkAttention>string</cbc:MarkAttention>
      <cbc:MarkCare>string</cbc:MarkCare>
      <cbc:PlotIdentification>string</cbc:PlotIdentification>
      <cbc:CitySubdivisionName>string</cbc:CitySubdivisionName>
      <cbc:CityName>string</cbc:CityName>
      <cbc:PostalZone>string</cbc:PostalZone>
      <cbc:CountrySubentity>string</cbc:CountrySubentity>

<cbc:CountrySubentityCode>normalizedString</cbc:CountrySubentityCod
e>
```

```
      <cbc:Region>string</cbc:Region>
      <cbc:District>string</cbc:District>
      <cbc:TimezoneOffset>string</cbc:TimezoneOffset>
      <cac:AddressLine>...          </cac:AddressLine>
      <cac:Country>...          </cac:Country>
      <cac:LocationCoordinate>...
</cac:LocationCoordinate>
    </cac:PostalAddress>
    <cac:PhysicalLocation>
      <cbc:ID>normalizedString</cbc:ID>
      <cbc:Description>string</cbc:Description>
      <cbc:Conditions>string</cbc:Conditions>
      <cbc:CountrySubentity>string</cbc:CountrySubentity>

<cbc:CountrySubentityCode>normalizedString</cbc:CountrySubentityCod
e>
      <cac:ValidityPeriod>...          </cac:ValidityPeriod>
      <cac:Address>...          </cac:Address>
    </cac:PhysicalLocation>
    <cac:PartyTaxScheme>
      <cbc:RegistrationName>string</cbc:RegistrationName>
      <cbc:CompanyID>normalizedString</cbc:CompanyID>
      <cbc:TaxLevelCode>normalizedString</cbc:TaxLevelCode>

<cbc:ExemptionReasonCode>normalizedString</cbc:ExemptionReasonCode>
      <cbc:ExemptionReason>string</cbc:ExemptionReason>
      <cac:RegistrationAddress>...
</cac:RegistrationAddress>
      <cac:TaxScheme>...          </cac:TaxScheme>
    </cac:PartyTaxScheme>
    <cac:PartyLegalEntity>
      <cbc:RegistrationName>string</cbc:RegistrationName>
      <cbc:CompanyID>normalizedString</cbc:CompanyID>
      <cac:RegistrationAddress>...
</cac:RegistrationAddress>
      <cac:CorporateRegistrationScheme>...
</cac:CorporateRegistrationScheme>
    </cac:PartyLegalEntity>
    <cac:Contact>
      <cbc:ID>normalizedString</cbc:ID>
      <cbc:Name>string</cbc:Name>
      <cbc:Telephone>string</cbc:Telephone>
      <cbc:Telefax>string</cbc:Telefax>
      <cbc:ElectronicMail>string</cbc:ElectronicMail>
      <cbc:Note>string</cbc:Note>
      <cac:OtherCommunication>...
</cac:OtherCommunication>
    </cac:Contact>
    <cac:Person>
      <cbc:FirstName>string</cbc:FirstName>
      <cbc:FamilyName>string</cbc:FamilyName>
      <cbc:Title>string</cbc:Title>
      <cbc:MiddleName>string</cbc:MiddleName>
      <cbc:NameSuffix>string</cbc:NameSuffix>
      <cbc:JobTitle>string</cbc:JobTitle>

<cbc:OrganizationDepartment>string</cbc:OrganizationDepartment>
    </cac:Person>
```

```
    <cac:AgentParty>
      <cbc:MarkCareIndicator>true</cbc:MarkCareIndicator>
      <cbc:MarkAttentionIndicator>true</cbc:MarkAttentionIndicator>
      <cbc:WebsiteURI>normalizedString</cbc:WebsiteURI>
      <cbc:LogoReferenceID>normalizedString</cbc:LogoReferenceID>
      <cbc:EndpointID>normalizedString</cbc:EndpointID>
      <cac:PartyIdentification>...
</cac:PartyIdentification>
      <cac:PartyName>...          </cac:PartyName>
      <cac:Language>...          </cac:Language>
      <cac:PostalAddress>...          </cac:PostalAddress>
      <cac:PhysicalLocation>...          </cac:PhysicalLocation>
      <cac:PartyTaxScheme>...          </cac:PartyTaxScheme>
      <cac:PartyLegalEntity>...          </cac:PartyLegalEntity>
      <cac:Contact>...          </cac:Contact>
      <cac:Person>...          </cac:Person>
      <cac:AgentParty>...
</cac:AgentParty></cac:AgentParty>
    </cac:FreightForwarderParty>
    <cac:AccountingCustomerParty>

<cbc:CustomerAssignedAccountID>normalizedString</cbc:CustomerAssign
edAccountID>

<cbc:SupplierAssignedAccountID>normalizedString</cbc:SupplierAssign
edAccountID>

<cbc:AdditionalAccountID>normalizedString</cbc:AdditionalAccountID>
      <cac:Party>
        <cbc:MarkCareIndicator>true</cbc:MarkCareIndicator>

<cbc:MarkAttentionIndicator>true</cbc:MarkAttentionIndicator>
        <cbc:WebsiteURI>normalizedString</cbc:WebsiteURI>
        <cbc:LogoReferenceID>normalizedString</cbc:LogoReferenceID>
        <cbc:EndpointID>normalizedString</cbc:EndpointID>
        <cac:PartyIdentification>...
</cac:PartyIdentification>
        <cac:PartyName>...          </cac:PartyName>
        <cac:Language>...          </cac:Language>
        <cac:PostalAddress>...          </cac:PostalAddress>
        <cac:PhysicalLocation>...          </cac:PhysicalLocation>
        <cac:PartyTaxScheme>...          </cac:PartyTaxScheme>
        <cac:PartyLegalEntity>...          </cac:PartyLegalEntity>
        <cac:Contact>...          </cac:Contact>
        <cac:Person>...          </cac:Person>
        <cac:AgentParty>...          </cac:AgentParty>
      </cac:Party>
      <cac:DeliveryContact>
        <cbc:ID>normalizedString</cbc:ID>
        <cbc:Name>string</cbc:Name>
        <cbc:Telephone>string</cbc:Telephone>
        <cbc:Telefax>string</cbc:Telefax>
        <cbc:ElectronicMail>string</cbc:ElectronicMail>
        <cbc:Note>string</cbc:Note>
        <cac:OtherCommunication>...
</cac:OtherCommunication>
      </cac:DeliveryContact>
      <cac:AccountingContact>
```

```
        <cbc:ID>normalizedString</cbc:ID>
        <cbc:Name>string</cbc:Name>
        <cbc:Telephone>string</cbc:Telephone>
        <cbc:Telefax>string</cbc:Telefax>
        <cbc:ElectronicMail>string</cbc:ElectronicMail>
        <cbc:Note>string</cbc:Note>
        <cac:OtherCommunication>...
</cac:OtherCommunication>
      </cac:AccountingContact>
      <cac:BuyerContact>
        <cbc:ID>normalizedString</cbc:ID>
        <cbc:Name>string</cbc:Name>
        <cbc:Telephone>string</cbc:Telephone>
        <cbc:Telefax>string</cbc:Telefax>
        <cbc:ElectronicMail>string</cbc:ElectronicMail>
        <cbc:Note>string</cbc:Note>
        <cac:OtherCommunication>...
</cac:OtherCommunication>
      </cac:BuyerContact>
    </cac:AccountingCustomerParty>
    <cac:AccountingSupplierParty>

<cbc:CustomerAssignedAccountID>normalizedString</cbc:CustomerAssign
edAccountID>

<cbc:AdditionalAccountID>normalizedString</cbc:AdditionalAccountID>
      <cbc:DataSendingCapability>string</cbc:DataSendingCapability>
      <cac:Party>
        <cbc:MarkCareIndicator>true</cbc:MarkCareIndicator>

<cbc:MarkAttentionIndicator>true</cbc:MarkAttentionIndicator>
        <cbc:WebsiteURI>normalizedString</cbc:WebsiteURI>
        <cbc:LogoReferenceID>normalizedString</cbc:LogoReferenceID>
        <cbc:EndpointID>normalizedString</cbc:EndpointID>
        <cac:PartyIdentification>...
</cac:PartyIdentification>
        <cac:PartyName>...          </cac:PartyName>
        <cac:Language>...          </cac:Language>
        <cac:PostalAddress>...          </cac:PostalAddress>
        <cac:PhysicalLocation>...          </cac:PhysicalLocation>
        <cac:PartyTaxScheme>...          </cac:PartyTaxScheme>
        <cac:PartyLegalEntity>...          </cac:PartyLegalEntity>
        <cac:Contact>...          </cac:Contact>
        <cac:Person>...          </cac:Person>
        <cac:AgentParty>...          </cac:AgentParty>
      </cac:Party>
      <cac:DespatchContact>
        <cbc:ID>normalizedString</cbc:ID>
        <cbc:Name>string</cbc:Name>
        <cbc:Telephone>string</cbc:Telephone>
        <cbc:Telefax>string</cbc:Telefax>
        <cbc:ElectronicMail>string</cbc:ElectronicMail>
        <cbc:Note>string</cbc:Note>
        <cac:OtherCommunication>...
</cac:OtherCommunication>
      </cac:DespatchContact>
      <cac:AccountingContact>
        <cbc:ID>normalizedString</cbc:ID>
```

```
        <cbc:Name>string</cbc:Name>
        <cbc:Telephone>string</cbc:Telephone>
        <cbc:Telefax>string</cbc:Telefax>
        <cbc:ElectronicMail>string</cbc:ElectronicMail>
        <cbc:Note>string</cbc:Note>
        <cac:OtherCommunication>...
</cac:OtherCommunication>
      </cac:AccountingContact>
      <cac:SellerContact>
        <cbc:ID>normalizedString</cbc:ID>
        <cbc:Name>string</cbc:Name>
        <cbc:Telephone>string</cbc:Telephone>
        <cbc:Telefax>string</cbc:Telefax>
        <cbc:ElectronicMail>string</cbc:ElectronicMail>
        <cbc:Note>string</cbc:Note>
        <cac:OtherCommunication>...
</cac:OtherCommunication>
      </cac:SellerContact>
    </cac:AccountingSupplierParty>
    <cac:Delivery>
      <cbc:ID>normalizedString</cbc:ID>
      <cbc:Quantity>1.0</cbc:Quantity>
      <cbc:MinimumQuantity>1.0</cbc:MinimumQuantity>
      <cbc:MaximumQuantity>1.0</cbc:MaximumQuantity>
      <cbc:ActualDeliveryDate>2000-01-01</cbc:ActualDeliveryDate>
      <cbc:ActualDeliveryTime>12:00:00</cbc:ActualDeliveryTime>
      <cbc:LatestDeliveryDate>2000-01-01</cbc:LatestDeliveryDate>
      <cbc:LatestDeliveryTime>12:00:00</cbc:LatestDeliveryTime>
      <cbc:TrackingID>normalizedString</cbc:TrackingID>
      <cac:DeliveryAddress>
        <cbc:ID>normalizedString</cbc:ID>
        <cbc:AddressTypeCode>normalizedString</cbc:AddressTypeCode>

<cbc:AddressFormatCode>normalizedString</cbc:AddressFormatCode>
        <cbc:Postbox>string</cbc:Postbox>
        <cbc:Floor>string</cbc:Floor>
        <cbc:Room>string</cbc:Room>
        <cbc:StreetName>string</cbc:StreetName>
        <cbc:AdditionalStreetName>string</cbc:AdditionalStreetName>
        <cbc:BlockName>string</cbc:BlockName>
        <cbc:BuildingName>string</cbc:BuildingName>
        <cbc:BuildingNumber>string</cbc:BuildingNumber>
        <cbc:InhouseMail>string</cbc:InhouseMail>
        <cbc:Department>string</cbc:Department>
        <cbc:MarkAttention>string</cbc:MarkAttention>
        <cbc:MarkCare>string</cbc:MarkCare>
        <cbc:PlotIdentification>string</cbc:PlotIdentification>
        <cbc:CitySubdivisionName>string</cbc:CitySubdivisionName>
        <cbc:CityName>string</cbc:CityName>
        <cbc:PostalZone>string</cbc:PostalZone>
        <cbc:CountrySubentity>string</cbc:CountrySubentity>

<cbc:CountrySubentityCode>normalizedString</cbc:CountrySubentityCod
e>
        <cbc:Region>string</cbc:Region>
        <cbc:District>string</cbc:District>
        <cbc:TimezoneOffset>string</cbc:TimezoneOffset>
        <cac:AddressLine>...          </cac:AddressLine>
```

```
        <cac:Country>...          </cac:Country>
        <cac:LocationCoordinate>...
</cac:LocationCoordinate>
      </cac:DeliveryAddress>
      <cac:DeliveryLocation>
        <cbc:ID>normalizedString</cbc:ID>
        <cbc:Description>string</cbc:Description>
        <cbc:Conditions>string</cbc:Conditions>
        <cbc:CountrySubentity>string</cbc:CountrySubentity>

<cbc:CountrySubentityCode>normalizedString</cbc:CountrySubentityCod
e>
        <cac:ValidityPeriod>...          </cac:ValidityPeriod>
        <cac:Address>...          </cac:Address>
      </cac:DeliveryLocation>
      <cac:RequestedDeliveryPeriod>
        <cbc:StartDate>2000-01-01</cbc:StartDate>
        <cbc:StartTime>12:00:00</cbc:StartTime>
        <cbc:EndDate>2000-01-01</cbc:EndDate>
        <cbc:EndTime>12:00:00</cbc:EndTime>
        <cbc:DurationMeasure
unitCode="04">1.0</cbc:DurationMeasure>
        <cbc:DescriptionCode>normalizedString</cbc:DescriptionCode>
        <cbc:Description>string</cbc:Description>
      </cac:RequestedDeliveryPeriod>
      <cac:PromisedDeliveryPeriod>
        <cbc:StartDate>2000-01-01</cbc:StartDate>
        <cbc:StartTime>12:00:00</cbc:StartTime>
        <cbc:EndDate>2000-01-01</cbc:EndDate>
        <cbc:EndTime>12:00:00</cbc:EndTime>
        <cbc:DurationMeasure
unitCode="04">1.0</cbc:DurationMeasure>
        <cbc:DescriptionCode>normalizedString</cbc:DescriptionCode>
        <cbc:Description>string</cbc:Description>
      </cac:PromisedDeliveryPeriod>
      <cac:EstimatedDeliveryPeriod>
        <cbc:StartDate>2000-01-01</cbc:StartDate>
        <cbc:StartTime>12:00:00</cbc:StartTime>
        <cbc:EndDate>2000-01-01</cbc:EndDate>
        <cbc:EndTime>12:00:00</cbc:EndTime>
        <cbc:DurationMeasure
unitCode="04">1.0</cbc:DurationMeasure>
        <cbc:DescriptionCode>normalizedString</cbc:DescriptionCode>
        <cbc:Description>string</cbc:Description>
      </cac:EstimatedDeliveryPeriod>
      <cac:DeliveryParty>
        <cbc:MarkCareIndicator>true</cbc:MarkCareIndicator>

<cbc:MarkAttentionIndicator>true</cbc:MarkAttentionIndicator>
        <cbc:WebsiteURI>normalizedString</cbc:WebsiteURI>
        <cbc:LogoReferenceID>normalizedString</cbc:LogoReferenceID>
        <cbc:EndpointID>normalizedString</cbc:EndpointID>
        <cac:PartyIdentification>...
</cac:PartyIdentification>
        <cac:PartyName>...          </cac:PartyName>
        <cac:Language>...          </cac:Language>
        <cac:PostalAddress>...          </cac:PostalAddress>
        <cac:PhysicalLocation>...          </cac:PhysicalLocation>
```

```
        <cac:PartyTaxScheme>...          </cac:PartyTaxScheme>
        <cac:PartyLegalEntity>...          </cac:PartyLegalEntity>
        <cac:Contact>...          </cac:Contact>
        <cac:Person>...          </cac:Person>
        <cac:AgentParty>...          </cac:AgentParty>
      </cac:DeliveryParty>
      <cac:Despatch>
        <cbc:ID>normalizedString</cbc:ID>
        <cbc:RequestedDespatchDate>2000-01-
01</cbc:RequestedDespatchDate>

<cbc:RequestedDespatchTime>12:00:00</cbc:RequestedDespatchTime>
        <cbc:EstimatedDespatchDate>2000-01-
01</cbc:EstimatedDespatchDate>

<cbc:EstimatedDespatchTime>12:00:00</cbc:EstimatedDespatchTime>
        <cbc:ActualDespatchDate>2000-01-01</cbc:ActualDespatchDate>
        <cbc:ActualDespatchTime>12:00:00</cbc:ActualDespatchTime>
        <cac:DespatchAddress>...          </cac:DespatchAddress>
        <cac:DespatchParty>...          </cac:DespatchParty>
        <cac:Contact>...          </cac:Contact>
      </cac:Despatch>
    </cac:Delivery>
    <cac:DeliveryTerms>
      <cbc:ID>normalizedString</cbc:ID>
      <cbc:SpecialTerms>string</cbc:SpecialTerms>

<cbc:LossRiskResponsibilityCode>normalizedString</cbc:LossRiskRespo
nsibilityCode>
      <cbc:LossRisk>string</cbc:LossRisk>
      <cac:DeliveryLocation>
        <cbc:ID>normalizedString</cbc:ID>
        <cbc:Description>string</cbc:Description>
        <cbc:Conditions>string</cbc:Conditions>
        <cbc:CountrySubentity>string</cbc:CountrySubentity>

<cbc:CountrySubentityCode>normalizedString</cbc:CountrySubentityCod
e>
        <cac:ValidityPeriod>...          </cac:ValidityPeriod>
        <cac:Address>...          </cac:Address>
      </cac:DeliveryLocation>
      <cac:AllowanceCharge>
        <cbc:ID>normalizedString</cbc:ID>
        <cbc:ChargeIndicator>true</cbc:ChargeIndicator>

<cbc:AllowanceChargeReasonCode>normalizedString</cbc:AllowanceCharg
eReasonCode>

<cbc:AllowanceChargeReason>string</cbc:AllowanceChargeReason>

<cbc:MultiplierFactorNumeric>1.0</cbc:MultiplierFactorNumeric>
        <cbc:PrepaidIndicator>true</cbc:PrepaidIndicator>
        <cbc:SequenceNumeric>1.0</cbc:SequenceNumeric>
        <cbc:Amount currencyID="AED">1.0</cbc:Amount>
        <cbc:BaseAmount currencyID="AED">1.0</cbc:BaseAmount>

<cbc:AccountingCostCode>normalizedString</cbc:AccountingCostCode>
        <cbc:AccountingCost>string</cbc:AccountingCost>
```

```
        <cac:TaxCategory>...          </cac:TaxCategory>
        <cac:TaxTotal>...          </cac:TaxTotal>
        <cac:PaymentMeans>...          </cac:PaymentMeans>
      </cac:AllowanceCharge>
    </cac:DeliveryTerms>
    <cac:PaymentMeans>
      <cbc:ID>normalizedString</cbc:ID>
      <cbc:PaymentMeansCode>normalizedString</cbc:PaymentMeansCode>
      <cbc:PaymentDueDate>2000-01-01</cbc:PaymentDueDate>

<cbc:PaymentChannelCode>normalizedString</cbc:PaymentChannelCode>
      <cbc:InstructionID>normalizedString</cbc:InstructionID>
      <cbc:InstructionNote>string</cbc:InstructionNote>
      <cbc:PaymentID>normalizedString</cbc:PaymentID>
      <cac:CardAccount>

<cbc:PrimaryAccountNumberID>normalizedString</cbc:PrimaryAccountNum
berID>
        <cbc:NetworkID>normalizedString</cbc:NetworkID>
        <cbc:CardTypeCode>normalizedString</cbc:CardTypeCode>
        <cbc:ValidityStartDate>2000-01-01</cbc:ValidityStartDate>
        <cbc:ExpiryDate>2000-01-01</cbc:ExpiryDate>
        <cbc:IssuerID>normalizedString</cbc:IssuerID>
        <cbc:IssueNumberID>normalizedString</cbc:IssueNumberID>
        <cbc:CV2ID>normalizedString</cbc:CV2ID>
        <cbc:CardChipCode>normalizedString</cbc:CardChipCode>

<cbc:ChipApplicationID>normalizedString</cbc:ChipApplicationID>
        <cbc:HolderName>string</cbc:HolderName>
      </cac:CardAccount>
      <cac:PayerFinancialAccount>
        <cbc:ID>normalizedString</cbc:ID>
        <cbc:Name>string</cbc:Name>
        <cbc:AccountTypeCode>normalizedString</cbc:AccountTypeCode>
        <cbc:CurrencyCode>normalizedString</cbc:CurrencyCode>
        <cbc:PaymentNote>string</cbc:PaymentNote>
        <cac:FinancialInstitutionBranch>...
</cac:FinancialInstitutionBranch>
        <cac:Country>...          </cac:Country>
      </cac:PayerFinancialAccount>
      <cac:PayeeFinancialAccount>
        <cbc:ID>normalizedString</cbc:ID>
        <cbc:Name>string</cbc:Name>
        <cbc:AccountTypeCode>normalizedString</cbc:AccountTypeCode>
        <cbc:CurrencyCode>normalizedString</cbc:CurrencyCode>
        <cbc:PaymentNote>string</cbc:PaymentNote>
        <cac:FinancialInstitutionBranch>...
</cac:FinancialInstitutionBranch>
        <cac:Country>...          </cac:Country>
      </cac:PayeeFinancialAccount>
      <cac:CreditAccount>
        <cbc:AccountID>normalizedString</cbc:AccountID>
      </cac:CreditAccount>
    </cac:PaymentMeans>
    <cac:TransactionConditions>
      <cbc:ID>normalizedString</cbc:ID>
      <cbc:ActionCode>normalizedString</cbc:ActionCode>
      <cbc:Description>string</cbc:Description>
```

```
      <cac:DocumentReference>
        <cbc:ID>normalizedString</cbc:ID>
        <cbc:CopyIndicator>true</cbc:CopyIndicator>
        <cbc:UUID>normalizedString</cbc:UUID>
        <cbc:IssueDate>2000-01-01</cbc:IssueDate>

<cbc:DocumentTypeCode>normalizedString</cbc:DocumentTypeCode>
        <cbc:DocumentType>string</cbc:DocumentType>
        <cbc:XPath>string</cbc:XPath>
        <cac:Attachment>...          </cac:Attachment>
      </cac:DocumentReference>
    </cac:TransactionConditions>
    <cac:AllowanceCharge>
      <cbc:ID>normalizedString</cbc:ID>
      <cbc:ChargeIndicator>true</cbc:ChargeIndicator>

<cbc:AllowanceChargeReasonCode>normalizedString</cbc:AllowanceCharg
eReasonCode>
      <cbc:AllowanceChargeReason>string</cbc:AllowanceChargeReason>

<cbc:MultiplierFactorNumeric>1.0</cbc:MultiplierFactorNumeric>
      <cbc:PrepaidIndicator>true</cbc:PrepaidIndicator>
      <cbc:SequenceNumeric>1.0</cbc:SequenceNumeric>
      <cbc:Amount currencyID="AED">1.0</cbc:Amount>
      <cbc:BaseAmount currencyID="AED">1.0</cbc:BaseAmount>

<cbc:AccountingCostCode>normalizedString</cbc:AccountingCostCode>
      <cbc:AccountingCost>string</cbc:AccountingCost>
      <cac:TaxCategory>
        <cbc:ID>normalizedString</cbc:ID>
        <cbc:Name>string</cbc:Name>
        <cbc:Percent>1.0</cbc:Percent>
        <cbc:BaseUnitMeasure
unitCode="04">1.0</cbc:BaseUnitMeasure>
        <cbc:PerUnitAmount currencyID="AED">1.0</cbc:PerUnitAmount>

<cbc:TaxExemptionReasonCode>normalizedString</cbc:TaxExemptionReaso
nCode>
        <cbc:TaxExemptionReason>string</cbc:TaxExemptionReason>
        <cbc:TierRange>string</cbc:TierRange>
        <cbc:TierRatePercent>1.0</cbc:TierRatePercent>
        <cac:TaxScheme>...          </cac:TaxScheme>
      </cac:TaxCategory>
      <cac:TaxTotal>
        <cbc:TaxAmount currencyID="AED">1.0</cbc:TaxAmount>
        <cbc:RoundingAmount
currencyID="AED">1.0</cbc:RoundingAmount>
        <cbc:TaxEvidenceIndicator>true</cbc:TaxEvidenceIndicator>
        <cac:TaxSubtotal>...          </cac:TaxSubtotal>
      </cac:TaxTotal>
      <cac:PaymentMeans>
        <cbc:ID>normalizedString</cbc:ID>

<cbc:PaymentMeansCode>normalizedString</cbc:PaymentMeansCode>
        <cbc:PaymentDueDate>2000-01-01</cbc:PaymentDueDate>

<cbc:PaymentChannelCode>normalizedString</cbc:PaymentChannelCode>
        <cbc:InstructionID>normalizedString</cbc:InstructionID>
```

```
        <cbc:InstructionNote>string</cbc:InstructionNote>
        <cbc:PaymentID>normalizedString</cbc:PaymentID>
        <cac:CardAccount>...          </cac:CardAccount>
        <cac:PayerFinancialAccount>...
</cac:PayerFinancialAccount>
        <cac:PayeeFinancialAccount>...
</cac:PayeeFinancialAccount>
        <cac:CreditAccount>...          </cac:CreditAccount>
      </cac:PaymentMeans>
    </cac:AllowanceCharge>
    <cac:DestinationCountry>

<cbc:IdentificationCode>normalizedString</cbc:IdentificationCode>
      <cbc:Name>string</cbc:Name>
    </cac:DestinationCountry>
    <cac:TaxTotal>
      <cbc:TaxAmount currencyID="AED">1.0</cbc:TaxAmount>
      <cbc:RoundingAmount currencyID="AED">1.0</cbc:RoundingAmount>
      <cbc:TaxEvidenceIndicator>true</cbc:TaxEvidenceIndicator>
      <cac:TaxSubtotal>
        <cbc:TaxableAmount currencyID="AED">1.0</cbc:TaxableAmount>
        <cbc:TaxAmount currencyID="AED">1.0</cbc:TaxAmount>

<cbc:CalculationSequenceNumeric>1.0</cbc:CalculationSequenceNumeric
>
        <cbc:TransactionCurrencyTaxAmount
currencyID="AED">1.0</cbc:TransactionCurrencyTaxAmount>
        <cbc:Percent>1.0</cbc:Percent>
        <cbc:BaseUnitMeasure
unitCode="04">1.0</cbc:BaseUnitMeasure>
        <cbc:PerUnitAmount currencyID="AED">1.0</cbc:PerUnitAmount>
        <cbc:TierRange>string</cbc:TierRange>
        <cbc:TierRatePercent>1.0</cbc:TierRatePercent>
        <cac:TaxCategory>...          </cac:TaxCategory>
      </cac:TaxSubtotal>
    </cac:TaxTotal>
    <cac:AnticipatedMonetaryTotal>
      <cbc:LineExtensionAmount
currencyID="AED">1.0</cbc:LineExtensionAmount>
      <cbc:TaxExclusiveAmount
currencyID="AED">1.0</cbc:TaxExclusiveAmount>
      <cbc:TaxInclusiveAmount
currencyID="AED">1.0</cbc:TaxInclusiveAmount>
      <cbc:AllowanceTotalAmount
currencyID="AED">1.0</cbc:AllowanceTotalAmount>
      <cbc:ChargeTotalAmount
currencyID="AED">1.0</cbc:ChargeTotalAmount>
      <cbc:PrepaidAmount currencyID="AED">1.0</cbc:PrepaidAmount>
      <cbc:PayableRoundingAmount
currencyID="AED">1.0</cbc:PayableRoundingAmount>
      <cbc:PayableAmount currencyID="AED">1.0</cbc:PayableAmount>
    </cac:AnticipatedMonetaryTotal>
    <cac:OrderLine>

<cbc:SubstitutionStatusCode>normalizedString</cbc:SubstitutionStatu
sCode>
      <cbc:Note>string</cbc:Note>
      <cac:LineItem>
```

```
        <cbc:ID>normalizedString</cbc:ID>
        <cbc:SalesOrderID>normalizedString</cbc:SalesOrderID>
        <cbc:UUID>normalizedString</cbc:UUID>
        <cbc:Note>string</cbc:Note>
        <cbc:LineStatusCode>normalizedString</cbc:LineStatusCode>
        <cbc:Quantity>1.0</cbc:Quantity>
        <cbc:LineExtensionAmount
currencyID="AED">1.0</cbc:LineExtensionAmount>
        <cbc:TotalTaxAmount
currencyID="AED">1.0</cbc:TotalTaxAmount>
        <cbc:MinimumQuantity>1.0</cbc:MinimumQuantity>
        <cbc:MaximumQuantity>1.0</cbc:MaximumQuantity>

<cbc:MinimumBackorderQuantity>1.0</cbc:MinimumBackorderQuantity>

<cbc:MaximumBackorderQuantity>1.0</cbc:MaximumBackorderQuantity>

<cbc:InspectionMethodCode>normalizedString</cbc:InspectionMethodCod
e>

<cbc:PartialDeliveryIndicator>true</cbc:PartialDeliveryIndicator>

<cbc:BackOrderAllowedIndicator>true</cbc:BackOrderAllowedIndicator>

<cbc:AccountingCostCode>normalizedString</cbc:AccountingCostCode>
        <cbc:AccountingCost>string</cbc:AccountingCost>
        <cac:Delivery>...          </cac:Delivery>
        <cac:DeliveryTerms>...          </cac:DeliveryTerms>
        <cac:OriginatorParty>...          </cac:OriginatorParty>
        <cac:OrderedShipment>...          </cac:OrderedShipment>
        <cac:PricingReference>...          </cac:PricingReference>
        <cac:AllowanceCharge>...          </cac:AllowanceCharge>
        <cac:Price>...          </cac:Price>
        <cac:Item>...          </cac:Item>
      </cac:LineItem>
      <cac:SellerProposedSubstituteLineItem>
        <cbc:ID>normalizedString</cbc:ID>
        <cbc:SalesOrderID>normalizedString</cbc:SalesOrderID>
        <cbc:UUID>normalizedString</cbc:UUID>
        <cbc:Note>string</cbc:Note>
        <cbc:LineStatusCode>normalizedString</cbc:LineStatusCode>
        <cbc:Quantity>1.0</cbc:Quantity>
        <cbc:LineExtensionAmount
currencyID="AED">1.0</cbc:LineExtensionAmount>
        <cbc:TotalTaxAmount
currencyID="AED">1.0</cbc:TotalTaxAmount>
        <cbc:MinimumQuantity>1.0</cbc:MinimumQuantity>
        <cbc:MaximumQuantity>1.0</cbc:MaximumQuantity>

<cbc:MinimumBackorderQuantity>1.0</cbc:MinimumBackorderQuantity>

<cbc:MaximumBackorderQuantity>1.0</cbc:MaximumBackorderQuantity>

<cbc:InspectionMethodCode>normalizedString</cbc:InspectionMethodCod
e>

<cbc:PartialDeliveryIndicator>true</cbc:PartialDeliveryIndicator>
```

```
<cbc:BackOrderAllowedIndicator>true</cbc:BackOrderAllowedIndicator>

<cbc:AccountingCostCode>normalizedString</cbc:AccountingCostCode>
        <cbc:AccountingCost>string</cbc:AccountingCost>
        <cac:Delivery>...          </cac:Delivery>
        <cac:DeliveryTerms>...          </cac:DeliveryTerms>
        <cac:OriginatorParty>...          </cac:OriginatorParty>
        <cac:OrderedShipment>...          </cac:OrderedShipment>
        <cac:PricingReference>...          </cac:PricingReference>
        <cac:AllowanceCharge>...          </cac:AllowanceCharge>
        <cac:Price>...          </cac:Price>
        <cac:Item>...          </cac:Item>
      </cac:SellerProposedSubstituteLineItem>
      <cac:SellerSubstitutedLineItem>
        <cbc:ID>normalizedString</cbc:ID>
        <cbc:SalesOrderID>normalizedString</cbc:SalesOrderID>
        <cbc:UUID>normalizedString</cbc:UUID>
        <cbc:Note>string</cbc:Note>
        <cbc:LineStatusCode>normalizedString</cbc:LineStatusCode>
        <cbc:Quantity>1.0</cbc:Quantity>
        <cbc:LineExtensionAmount
currencyID="AED">1.0</cbc:LineExtensionAmount>
        <cbc:TotalTaxAmount
currencyID="AED">1.0</cbc:TotalTaxAmount>
        <cbc:MinimumQuantity>1.0</cbc:MinimumQuantity>
        <cbc:MaximumQuantity>1.0</cbc:MaximumQuantity>

<cbc:MinimumBackorderQuantity>1.0</cbc:MinimumBackorderQuantity>

<cbc:MaximumBackorderQuantity>1.0</cbc:MaximumBackorderQuantity>

<cbc:InspectionMethodCode>normalizedString</cbc:InspectionMethodCod
e>

<cbc:PartialDeliveryIndicator>true</cbc:PartialDeliveryIndicator>

<cbc:BackOrderAllowedIndicator>true</cbc:BackOrderAllowedIndicator>

<cbc:AccountingCostCode>normalizedString</cbc:AccountingCostCode>
        <cbc:AccountingCost>string</cbc:AccountingCost>
        <cac:Delivery>...          </cac:Delivery>
        <cac:DeliveryTerms>...          </cac:DeliveryTerms>
        <cac:OriginatorParty>...          </cac:OriginatorParty>
        <cac:OrderedShipment>...          </cac:OrderedShipment>
        <cac:PricingReference>...          </cac:PricingReference>
        <cac:AllowanceCharge>...          </cac:AllowanceCharge>
        <cac:Price>...          </cac:Price>
        <cac:Item>...          </cac:Item>
      </cac:SellerSubstitutedLineItem>
      <cac:BuyerProposedSubstituteLineItem>
        <cbc:ID>normalizedString</cbc:ID>
        <cbc:SalesOrderID>normalizedString</cbc:SalesOrderID>
        <cbc:UUID>normalizedString</cbc:UUID>
        <cbc:Note>string</cbc:Note>
        <cbc:LineStatusCode>normalizedString</cbc:LineStatusCode>
        <cbc:Quantity>1.0</cbc:Quantity>
```

```
        <cbc:LineExtensionAmount
currencyID="AED">1.0</cbc:LineExtensionAmount>
        <cbc:TotalTaxAmount
currencyID="AED">1.0</cbc:TotalTaxAmount>
        <cbc:MinimumQuantity>1.0</cbc:MinimumQuantity>
        <cbc:MaximumQuantity>1.0</cbc:MaximumQuantity>

<cbc:MinimumBackorderQuantity>1.0</cbc:MinimumBackorderQuantity>

<cbc:MaximumBackorderQuantity>1.0</cbc:MaximumBackorderQuantity>

<cbc:InspectionMethodCode>normalizedString</cbc:InspectionMethodCod
e>

<cbc:PartialDeliveryIndicator>true</cbc:PartialDeliveryIndicator>

<cbc:BackOrderAllowedIndicator>true</cbc:BackOrderAllowedIndicator>

<cbc:AccountingCostCode>normalizedString</cbc:AccountingCostCode>
        <cbc:AccountingCost>string</cbc:AccountingCost>
        <cac:Delivery>...          </cac:Delivery>
        <cac:DeliveryTerms>...          </cac:DeliveryTerms>
        <cac:OriginatorParty>...          </cac:OriginatorParty>
        <cac:OrderedShipment>...          </cac:OrderedShipment>
        <cac:PricingReference>...          </cac:PricingReference>
        <cac:AllowanceCharge>...          </cac:AllowanceCharge>
        <cac:Price>...          </cac:Price>
        <cac:Item>...          </cac:Item>
      </cac:BuyerProposedSubstituteLineItem>
      <cac:CatalogueLineReference>
        <cbc:LineID>normalizedString</cbc:LineID>
        <cbc:UUID>normalizedString</cbc:UUID>
        <cbc:LineStatusCode>normalizedString</cbc:LineStatusCode>
        <cac:DocumentReference>...
</cac:DocumentReference>
      </cac:CatalogueLineReference>
      <cac:QuotationLineReference>
        <cbc:LineID>normalizedString</cbc:LineID>
        <cbc:UUID>normalizedString</cbc:UUID>
        <cbc:LineStatusCode>normalizedString</cbc:LineStatusCode>
        <cac:DocumentReference>...
</cac:DocumentReference>
      </cac:QuotationLineReference>
      <cac:DocumentReference>
        <cbc:ID>normalizedString</cbc:ID>
        <cbc:CopyIndicator>true</cbc:CopyIndicator>
        <cbc:UUID>normalizedString</cbc:UUID>
        <cbc:IssueDate>2000-01-01</cbc:IssueDate>

<cbc:DocumentTypeCode>normalizedString</cbc:DocumentTypeCode>
        <cbc:DocumentType>string</cbc:DocumentType>
        <cbc:XPath>string</cbc:XPath>
        <cac:Attachment>...          </cac:Attachment>
      </cac:DocumentReference>
    </cac:OrderLine>
  </OrderChange>
```

Appendix 17: Sample UBL 2.0 Purchase Order Cancellation example (Source: OAGIS)

```
<OrderCancellation>
  <ext:UBLExtensions>
    <ext:UBLExtension>
      <cbc:ID>normalizedString</cbc:ID>
      <cbc:Name>string</cbc:Name>

<ext:ExtensionAgencyID>normalizedString</ext:ExtensionAgencyID>
      <ext:ExtensionAgencyName>string</ext:ExtensionAgencyName>

<ext:ExtensionVersionID>normalizedString</ext:ExtensionVersionID>

<ext:ExtensionAgencyURI>normalizedString</ext:ExtensionAgencyURI>
      <ext:ExtensionURI>normalizedString</ext:ExtensionURI>

<ext:ExtensionReasonCode>normalizedString</ext:ExtensionReasonCode>
      <ext:ExtensionReason>string</ext:ExtensionReason>
      <ext:ExtensionContent>
        <!--any element-->
      </ext:ExtensionContent>
    </ext:UBLExtension>
  </ext:UBLExtensions>
  <cbc:UBLVersionID>normalizedString</cbc:UBLVersionID>
  <cbc:CustomizationID>normalizedString</cbc:CustomizationID>
  <cbc:ProfileID>normalizedString</cbc:ProfileID>
  <cbc:ID>normalizedString</cbc:ID>
  <cbc:CopyIndicator>true</cbc:CopyIndicator>
  <cbc:UUID>normalizedString</cbc:UUID>
  <cbc:IssueDate>2000-01-01</cbc:IssueDate>
  <cbc:IssueTime>12:00:00</cbc:IssueTime>
  <cbc:Note>string</cbc:Note>
  <cbc:CancellationNote>string</cbc:CancellationNote>
  <cac:OrderReference>
    <cbc:ID>normalizedString</cbc:ID>
    <cbc:SalesOrderID>normalizedString</cbc:SalesOrderID>
    <cbc:CopyIndicator>true</cbc:CopyIndicator>
    <cbc:UUID>normalizedString</cbc:UUID>
    <cbc:IssueDate>2000-01-01</cbc:IssueDate>
    <cbc:IssueTime>12:00:00</cbc:IssueTime>
    <cbc:CustomerReference>string</cbc:CustomerReference>
    <cac:DocumentReference>
      <cbc:ID>normalizedString</cbc:ID>
      <cbc:CopyIndicator>true</cbc:CopyIndicator>
      <cbc:UUID>normalizedString</cbc:UUID>
      <cbc:IssueDate>2000-01-01</cbc:IssueDate>
      <cbc:DocumentTypeCode>normalizedString</cbc:DocumentTypeCode>
      <cbc:DocumentType>string</cbc:DocumentType>
      <cbc:XPath>string</cbc:XPath>
      <cac:Attachment>...          </cac:Attachment>
    </cac:DocumentReference>
  </cac:OrderReference>
  <cac:OriginatorDocumentReference>
    <cbc:ID>normalizedString</cbc:ID>
    <cbc:CopyIndicator>true</cbc:CopyIndicator>
    <cbc:UUID>normalizedString</cbc:UUID>
```

```
    <cbc:IssueDate>2000-01-01</cbc:IssueDate>
    <cbc:DocumentTypeCode>normalizedString</cbc:DocumentTypeCode>
    <cbc:DocumentType>string</cbc:DocumentType>
    <cbc:XPath>string</cbc:XPath>
    <cac:Attachment>
      <cbc:EmbeddedDocumentBinaryObject
mimeCode="application/CSTAdata+xml">GpM7</cbc:EmbeddedDocumentBinar
yObject>
      <cac:ExternalReference>...          </cac:ExternalReference>
    </cac:Attachment>
  </cac:OriginatorDocumentReference>
  <cac:AdditionalDocumentReference>
    <cbc:ID>normalizedString</cbc:ID>
    <cbc:CopyIndicator>true</cbc:CopyIndicator>
    <cbc:UUID>normalizedString</cbc:UUID>
    <cbc:IssueDate>2000-01-01</cbc:IssueDate>
    <cbc:DocumentTypeCode>normalizedString</cbc:DocumentTypeCode>
    <cbc:DocumentType>string</cbc:DocumentType>
    <cbc:XPath>string</cbc:XPath>
    <cac:Attachment>
      <cbc:EmbeddedDocumentBinaryObject
mimeCode="application/CSTAdata+xml">GpM7</cbc:EmbeddedDocumentBinar
yObject>
      <cac:ExternalReference>...          </cac:ExternalReference>
    </cac:Attachment>
  </cac:AdditionalDocumentReference>
  <cac:Contract>
    <cbc:ID>normalizedString</cbc:ID>
    <cbc:IssueDate>2000-01-01</cbc:IssueDate>
    <cbc:IssueTime>12:00:00</cbc:IssueTime>
    <cbc:ContractTypeCode>normalizedString</cbc:ContractTypeCode>
    <cbc:ContractType>string</cbc:ContractType>
    <cac:ValidityPeriod>
      <cbc:StartDate>2000-01-01</cbc:StartDate>
      <cbc:StartTime>12:00:00</cbc:StartTime>
      <cbc:EndDate>2000-01-01</cbc:EndDate>
      <cbc:EndTime>12:00:00</cbc:EndTime>
      <cbc:DurationMeasure unitCode="04">1.0</cbc:DurationMeasure>
      <cbc:DescriptionCode>normalizedString</cbc:DescriptionCode>
      <cbc:Description>string</cbc:Description>
    </cac:ValidityPeriod>
    <cac:ContractDocumentReference>
      <cbc:ID>normalizedString</cbc:ID>
      <cbc:CopyIndicator>true</cbc:CopyIndicator>
      <cbc:UUID>normalizedString</cbc:UUID>
      <cbc:IssueDate>2000-01-01</cbc:IssueDate>
      <cbc:DocumentTypeCode>normalizedString</cbc:DocumentTypeCode>
      <cbc:DocumentType>string</cbc:DocumentType>
      <cbc:XPath>string</cbc:XPath>
      <cac:Attachment>...          </cac:Attachment>
    </cac:ContractDocumentReference>
  </cac:Contract>
  <cac:Signature>
    <cbc:ID>normalizedString</cbc:ID>
    <cbc:Note>string</cbc:Note>
    <cbc:ValidationDate>2000-01-01</cbc:ValidationDate>
    <cbc:ValidationTime>12:00:00</cbc:ValidationTime>
    <cbc:ValidatorID>normalizedString</cbc:ValidatorID>
```

```
    <cbc:CanonicalizationMethod>string</cbc:CanonicalizationMethod>
    <cbc:SignatureMethod>string</cbc:SignatureMethod>
    <cac:SignatoryParty>
      <cbc:MarkCareIndicator>true</cbc:MarkCareIndicator>
      <cbc:MarkAttentionIndicator>true</cbc:MarkAttentionIndicator>
      <cbc:WebsiteURI>normalizedString</cbc:WebsiteURI>
      <cbc:LogoReferenceID>normalizedString</cbc:LogoReferenceID>
      <cbc:EndpointID>normalizedString</cbc:EndpointID>
      <cac:PartyIdentification>...
</cac:PartyIdentification>
      <cac:PartyName>...          </cac:PartyName>
      <cac:Language>...          </cac:Language>
      <cac:PostalAddress>...          </cac:PostalAddress>
      <cac:PhysicalLocation>...          </cac:PhysicalLocation>
      <cac:PartyTaxScheme>...          </cac:PartyTaxScheme>
      <cac:PartyLegalEntity>...          </cac:PartyLegalEntity>
      <cac:Contact>...          </cac:Contact>
      <cac:Person>...          </cac:Person>
      <cac:AgentParty>...          </cac:AgentParty>
    </cac:SignatoryParty>
    <cac:DigitalSignatureAttachment>
      <cbc:EmbeddedDocumentBinaryObject
mimeCode="application/CSTAdata+xml">GpM7</cbc:EmbeddedDocumentBinar
yObject>
      <cac:ExternalReference>...          </cac:ExternalReference>
    </cac:DigitalSignatureAttachment>
    <cac:OriginalDocumentReference>
      <cbc:ID>normalizedString</cbc:ID>
      <cbc:CopyIndicator>true</cbc:CopyIndicator>
      <cbc:UUID>normalizedString</cbc:UUID>
      <cbc:IssueDate>2000-01-01</cbc:IssueDate>
      <cbc:DocumentTypeCode>normalizedString</cbc:DocumentTypeCode>
      <cbc:DocumentType>string</cbc:DocumentType>
      <cbc:XPath>string</cbc:XPath>
      <cac:Attachment>...          </cac:Attachment>
    </cac:OriginalDocumentReference>
  </cac:Signature>
  <cac:BuyerCustomerParty>

<cbc:CustomerAssignedAccountID>normalizedString</cbc:CustomerAssign
edAccountID>

<cbc:SupplierAssignedAccountID>normalizedString</cbc:SupplierAssign
edAccountID>

<cbc:AdditionalAccountID>normalizedString</cbc:AdditionalAccountID>
    <cac:Party>
      <cbc:MarkCareIndicator>true</cbc:MarkCareIndicator>
      <cbc:MarkAttentionIndicator>true</cbc:MarkAttentionIndicator>
      <cbc:WebsiteURI>normalizedString</cbc:WebsiteURI>
      <cbc:LogoReferenceID>normalizedString</cbc:LogoReferenceID>
      <cbc:EndpointID>normalizedString</cbc:EndpointID>
      <cac:PartyIdentification>...
</cac:PartyIdentification>
      <cac:PartyName>...          </cac:PartyName>
      <cac:Language>...          </cac:Language>
      <cac:PostalAddress>...          </cac:PostalAddress>
      <cac:PhysicalLocation>...          </cac:PhysicalLocation>
```

```
      <cac:PartyTaxScheme>...          </cac:PartyTaxScheme>
      <cac:PartyLegalEntity>...          </cac:PartyLegalEntity>
      <cac:Contact>...          </cac:Contact>
      <cac:Person>...          </cac:Person>
      <cac:AgentParty>...          </cac:AgentParty>
    </cac:Party>
    <cac:DeliveryContact>
      <cbc:ID>normalizedString</cbc:ID>
      <cbc:Name>string</cbc:Name>
      <cbc:Telephone>string</cbc:Telephone>
      <cbc:Telefax>string</cbc:Telefax>
      <cbc:ElectronicMail>string</cbc:ElectronicMail>
      <cbc:Note>string</cbc:Note>
      <cac:OtherCommunication>...
</cac:OtherCommunication>
    </cac:DeliveryContact>
    <cac:AccountingContact>
      <cbc:ID>normalizedString</cbc:ID>
      <cbc:Name>string</cbc:Name>
      <cbc:Telephone>string</cbc:Telephone>
      <cbc:Telefax>string</cbc:Telefax>
      <cbc:ElectronicMail>string</cbc:ElectronicMail>
      <cbc:Note>string</cbc:Note>
      <cac:OtherCommunication>...
</cac:OtherCommunication>
    </cac:AccountingContact>
    <cac:BuyerContact>
      <cbc:ID>normalizedString</cbc:ID>
      <cbc:Name>string</cbc:Name>
      <cbc:Telephone>string</cbc:Telephone>
      <cbc:Telefax>string</cbc:Telefax>
      <cbc:ElectronicMail>string</cbc:ElectronicMail>
      <cbc:Note>string</cbc:Note>
      <cac:OtherCommunication>...
</cac:OtherCommunication>
    </cac:BuyerContact>
  </cac:BuyerCustomerParty>
  <cac:SellerSupplierParty>

<cbc:CustomerAssignedAccountID>normalizedString</cbc:CustomerAssign
edAccountID>

<cbc:AdditionalAccountID>normalizedString</cbc:AdditionalAccountID>
    <cbc:DataSendingCapability>string</cbc:DataSendingCapability>
    <cac:Party>
      <cbc:MarkCareIndicator>true</cbc:MarkCareIndicator>
      <cbc:MarkAttentionIndicator>true</cbc:MarkAttentionIndicator>
      <cbc:WebsiteURI>normalizedString</cbc:WebsiteURI>
      <cbc:LogoReferenceID>normalizedString</cbc:LogoReferenceID>
      <cbc:EndpointID>normalizedString</cbc:EndpointID>
      <cac:PartyIdentification>...
</cac:PartyIdentification>
      <cac:PartyName>...          </cac:PartyName>
      <cac:Language>...          </cac:Language>
      <cac:PostalAddress>...          </cac:PostalAddress>
      <cac:PhysicalLocation>...          </cac:PhysicalLocation>
      <cac:PartyTaxScheme>...          </cac:PartyTaxScheme>
      <cac:PartyLegalEntity>...          </cac:PartyLegalEntity>
```

```
      <cac:Contact>...          </cac:Contact>
      <cac:Person>...          </cac:Person>
      <cac:AgentParty>...          </cac:AgentParty>
    </cac:Party>
    <cac:DespatchContact>
      <cbc:ID>normalizedString</cbc:ID>
      <cbc:Name>string</cbc:Name>
      <cbc:Telephone>string</cbc:Telephone>
      <cbc:Telefax>string</cbc:Telefax>
      <cbc:ElectronicMail>string</cbc:ElectronicMail>
      <cbc:Note>string</cbc:Note>
      <cac:OtherCommunication>...
</cac:OtherCommunication>
    </cac:DespatchContact>
    <cac:AccountingContact>
      <cbc:ID>normalizedString</cbc:ID>
      <cbc:Name>string</cbc:Name>
      <cbc:Telephone>string</cbc:Telephone>
      <cbc:Telefax>string</cbc:Telefax>
      <cbc:ElectronicMail>string</cbc:ElectronicMail>
      <cbc:Note>string</cbc:Note>
      <cac:OtherCommunication>...
</cac:OtherCommunication>
    </cac:AccountingContact>
    <cac:SellerContact>
      <cbc:ID>normalizedString</cbc:ID>
      <cbc:Name>string</cbc:Name>
      <cbc:Telephone>string</cbc:Telephone>
      <cbc:Telefax>string</cbc:Telefax>
      <cbc:ElectronicMail>string</cbc:ElectronicMail>
      <cbc:Note>string</cbc:Note>
      <cac:OtherCommunication>...
</cac:OtherCommunication>
    </cac:SellerContact>
  </cac:SellerSupplierParty>
  <cac:OriginatorCustomerParty>

<cbc:CustomerAssignedAccountID>normalizedString</cbc:CustomerAssign
edAccountID>

<cbc:SupplierAssignedAccountID>normalizedString</cbc:SupplierAssign
edAccountID>

<cbc:AdditionalAccountID>normalizedString</cbc:AdditionalAccountID>
    <cac:Party>
      <cbc:MarkCareIndicator>true</cbc:MarkCareIndicator>
      <cbc:MarkAttentionIndicator>true</cbc:MarkAttentionIndicator>
      <cbc:WebsiteURI>normalizedString</cbc:WebsiteURI>
      <cbc:LogoReferenceID>normalizedString</cbc:LogoReferenceID>
      <cbc:EndpointID>normalizedString</cbc:EndpointID>
      <cac:PartyIdentification>...
</cac:PartyIdentification>
      <cac:PartyName>...          </cac:PartyName>
      <cac:Language>...          </cac:Language>
      <cac:PostalAddress>...          </cac:PostalAddress>
      <cac:PhysicalLocation>...          </cac:PhysicalLocation>
      <cac:PartyTaxScheme>...          </cac:PartyTaxScheme>
      <cac:PartyLegalEntity>...          </cac:PartyLegalEntity>
```

```
      <cac:Contact>...          </cac:Contact>
      <cac:Person>...          </cac:Person>
      <cac:AgentParty>...          </cac:AgentParty>
    </cac:Party>
    <cac:DeliveryContact>
      <cbc:ID>normalizedString</cbc:ID>
      <cbc:Name>string</cbc:Name>
      <cbc:Telephone>string</cbc:Telephone>
      <cbc:Telefax>string</cbc:Telefax>
      <cbc:ElectronicMail>string</cbc:ElectronicMail>
      <cbc:Note>string</cbc:Note>
      <cac:OtherCommunication>...
</cac:OtherCommunication>
    </cac:DeliveryContact>
    <cac:AccountingContact>
      <cbc:ID>normalizedString</cbc:ID>
      <cbc:Name>string</cbc:Name>
      <cbc:Telephone>string</cbc:Telephone>
      <cbc:Telefax>string</cbc:Telefax>
      <cbc:ElectronicMail>string</cbc:ElectronicMail>
      <cbc:Note>string</cbc:Note>
      <cac:OtherCommunication>...
</cac:OtherCommunication>
    </cac:AccountingContact>
    <cac:BuyerContact>
      <cbc:ID>normalizedString</cbc:ID>
      <cbc:Name>string</cbc:Name>
      <cbc:Telephone>string</cbc:Telephone>
      <cbc:Telefax>string</cbc:Telefax>
      <cbc:ElectronicMail>string</cbc:ElectronicMail>
      <cbc:Note>string</cbc:Note>
      <cac:OtherCommunication>...
</cac:OtherCommunication>
    </cac:BuyerContact>
  </cac:OriginatorCustomerParty>
</OrderCancellation>
```