XML-based Frameworks for Internet Commerce and an Implementation of B2B e-procurement

by

Yuxiao Zhao

Submitted to the School of Engineering at Linköping University in partial fulfilment of the requirements for the degree of Licentiate of Technology

Department of Computer and Information Science
Linköpings universitet
SE-581 83 Linköping, Sweden

Linköping 2001
XML-based Frameworks for Internet Commerce and an Implementation of B2B e-procurement

by

Yuxiao Zhao

April 2001
ISBN 91-7373-013-0
Linköpings Studies in Science and Technology
Thesis No. 882
ISSN 0280-7971
LiU-Tek-Lic-2001:19

ABSTRACT

It is not easy to apply XML in e-commerce development for achieving interoperability in heterogeneous environments. One of the reasons is a multitude of XML-based Frameworks for Internet Commerce (XFIC), or industrial standards. This thesis surveys 15 frameworks, i.e., ebXML, eCo Framework, UDDI, SOAP, BizTalk, cXML, ICE, Open Applications Group, RosettaNet, Wf-XML, OFX, VoiceXML, RDF, WSDL and xCBL.

This thesis provides three models to systematically understand how the 15 frameworks meet the requirements of e-commerce. A hierarchical model is presented to show the purpose and focus of various XFIC initiatives. A relationship model is given to show the cooperative and competitive relationships between XFIC. A chronological model is provided to look at the development of XFIC. In addition, the thesis offers guidelines for how to apply XFIC in an e-commerce development.

We have also implemented a B2B e-procurement system. That not only demonstrates the feasibility of open-source or freeware, but also validates the complementary roles of XML and Java: XML is for describing contents and Java is for automating XML documents (session handling). Auction-based dynamic pricing is also realized as a feature of interest. Moreover, the implementation shows the suitability of e-procurement for educational purposes in e-commerce development.

This work has been supported by SSF (Swedish Foundation for Strategic Research) through ECSEL (Excellence Center in Computer Science and Systems Engineering in Linköping).
To Lin
Acknowledgments

I have received a great number of helps from a lot of people, since I became a Ph.D student of STEM (Software Technology and Methodology), ECSEL (Excellence Center in Computer Science and Systems Engineering in Linköping), affiliated with PELAB (Programming Environment Laboratory), Dept of Computer and Information Science (IDA), Linköping University, Sweden.

First of all, I wish to thank deeply my supervisor, Prof. Kristian Sandahl, for his invaluable discussions, numerous industrial contacts, meaningful research suggestions, repetitive thesis enhancements, and so on. Without his support, this thesis would be impossible.

I am indebted to Prof. Henrik Eriksson for his very interesting comments that have greatly improved this thesis.

My special thanks go to Dr. Mary E. Helander, for her recommendation to become a graduate student at IDA and encouragement and suggestions during my study.

I am also very grateful to all people at IDA, in particular, Prof. Peter Fritzson for PELAB management; Prof. Mariam Kamkar for STEM management; and Prof. Nahid Shahmehri for ECSEL management; Bodil Mattsson Kihlström and Gunilla Mellhegen for the daily administration, and Lillemor Wallgren for thesis publications.

Three industrial professionals discussed with me about my project, including Christer Söderlund from Intentia Consulting at Linköping, Pär Wetterlöf from Industri-Matematic International (IMI), and Fredick Jansson from IFS AB. Thanks for your tactical suggestions.

Ivan Rankin and Angela Yong help improve the English language of this thesis. Thank you very much.

Lastly, I would like to thank all family members for their spiritual support, in particular, my wife, Lin Han, for her interesting comments and countless supports in life.

This work has been supported by the Swedish Foundation for Strategic Research through ECSEL. Thanks for the financial aids.

Yuxiao Zhao

Linköping, April 2001
# CONTENTS

1 INTRODUCTION.................................................................................................................. 1

1.1 SITUATION .................................................................................................................. 1

1.1.1 e-business and e-commerce ................................................................................. 1

1.1.2 e-procurement ..................................................................................................... 2

1.1.3 XML ....................................................................................................................... 3

1.2 COMPLICATION ......................................................................................................... 5

1.3 CONTRIBUTIONS ....................................................................................................... 5

1.4 THE ORGANIZATION OF THE THESIS ..................................................................... 6

2 XML-BASED FRAMEWORKS FOR INTERNET COMMERCE (XFIC)............ 7

2.1 INTRODUCTION ......................................................................................................... 7

2.1.1 Motivation ........................................................................................................... 7

2.1.2 Selection criteria ................................................................................................. 7

2.1.3 Related work ........................................................................................................ 10

2.2 FROM XML TO XFIC ............................................................................................... 11

2.2.1 XML Applications ............................................................................................. 11

2.2.2 The roles of XML in Internet commerce: interoperability ................................. 11

2.2.3 A generic model for XFIC .................................................................................. 14

2.2.4 Philosophy behind XFIC .................................................................................... 15

2.2.5 XFIC and Web services ...................................................................................... 16

2.3 THREE MODELS FOR DESCRIBING XFIC ........................................................... 18

2.3.1 Business processes for Internet commerce ....................................................... 18

2.3.2 A Hierarchical Model ........................................................................................ 20

2.3.3 A Relationship Model ....................................................................................... 26

2.3.4 A Chronological Model ..................................................................................... 28

3 XFIC: FOCUS ON THE BIG PICTURE ................................................................. 31

3.1 EBXML ...................................................................................................................... 31

3.1.1 System overview ............................................................................................... 31

3.1.2 Business Operational View and Functional Service View .............................. 33

3.1.3 Registry and Repository ................................................................................... 33

3.1.4 CPP and Collaboration Protocol Agreement .................................................. 34

3.1.5 Modeling business interactions ....................................................................... 35

3.1.6 Messaging Service ............................................................................................. 36

3.2 ECO FRAMEWORK ................................................................................................. 44

3.3 UDDI ......................................................................................................................... 47

3.3.1 Overview ............................................................................................................ 47

3.3.2 Registry Operator ............................................................................................... 47
3.3.3 Cost to register and assess to the Registry Operator ........................................ 47
3.3.4 UDDI APIs ........................................................................................................ 47
3.4 A COMPARISON OF THE FRAMEWORKS .......................................................... 52
  3.4.1 Main objective ................................................................................................ 52
  3.4.2 Architecture .................................................................................................... 52
  3.4.3 Registry in ebXML and UDDI ....................................................................... 52
  3.4.4 Marketing strategy ........................................................................................ 53
  3.4.5 Disadvantages ............................................................................................... 54
  3.4.6 Maturity .......................................................................................................... 54
3.5 FUTURE DEVELOPMENT .................................................................................... 55

4 XFIC: FOCUS ON XML MESSAGING .................................................................... 57
  4.1 SOAP AND BIZTALK ...................................................................................... 57
  4.1.1 SOAP (Simple Object Access Protocol) ....................................................... 57
  4.1.2 BizTalk ......................................................................................................... 59
  4.2 SPECIFIC BUSINESS OPERATIONS .............................................................. 63
    4.2.1 cXML (commerce XML) ......................................................................... 63
    4.2.2 ICE (Information and Content Exchange) ............................................. 66
  4.3 ENTERPRISE APPLICATION INTEGRATION (EAI) ........................................ 72
    4.3.1 OAG (Open Applications Group) .............................................................. 72
    4.3.2 RosettaNet .................................................................................................. 75
    4.3.3 Wf-XML (Workflow XML) ..................................................................... 77
  4.4 VERTICAL INDUSTRY ..................................................................................... 81
    4.4.1 OFX (Open Financial eXchange) .............................................................. 81
  4.5 SPECIFIC APPLICATIONS ............................................................................... 82
    4.5.1 VoiceXML .................................................................................................. 82
  4.6 SYNTAX AND SEMANTICS .............................................................................. 84
    4.6.1 WSDL (Web Services Description Language) ....................................... 84
    4.6.2 RDF (Resource Description Framework) ............................................. 87
  4.7 VOCABULARY .................................................................................................. 88
    4.7.1 xCBL (XML Common Business Library) .......................................... 88

5 AN IMPLEMENTATION OF B2B E-PROCUREMENT ........................................ 89
  5.1 MOTIVATION .................................................................................................... 89
  5.2 REQUIREMENTS CAPTURE ............................................................................ 90
    5.2.1 Inputs and outputs ..................................................................................... 90
    5.2.2 Assumptions .............................................................................................. 90
    5.2.3 Requirements ........................................................................................... 91
  5.3 BUSINESS USE CASES .................................................................................. 93
    5.3.1 The role of Customer and their use cases ............................................ 95
    5.3.2 The role of Timer and its use cases ....................................................... 96
  5.4 ANALYSIS AND DESIGN .............................................................................. 100
    5.4.1 Class diagram for the whole system ..................................................... 100
    5.4.2 Detailed Classes ...................................................................................... 103
5.5 IMPLEMENTATION PROCESSES USING JAVA AND XML ................................. 106
  5.5.1 Identifying important components......................................................... 106
  5.5.2 Practical hints......................................................................................... 108
5.6 TESTING........................................................................................................ 109
  5.6.1 Unit testing............................................................................................... 109
  5.6.2 Use case testing....................................................................................... 109
  5.6.3 System testing.......................................................................................... 110
5.7 DISCUSSION.................................................................................................... 114
  5.7.1 Open source or freeware in developing B2B e-commerce...................... 114
  5.7.2 Designing a suitable project for e-commerce training............................. 115
  5.7.3 XML and Java are complementary......................................................... 115
  5.7.4 Dynamic pricing based on auction.......................................................... 118
  5.7.5 XFIC and e-procurement implementation............................................. 119

6 SUMMARY AND CONCLUSIONS ................................................................. 121
  6.1 WHAT XFIC ARE......................................................................................... 121
  6.2 HOW XFIC INTERACT................................................................................ 121
  6.3 AN IMPLEMENTATION OF B2B E-PROCUREMENT............................... 122
  6.4 APPLYING XFIC IN E-COMMERCE DEVELOPMENT................................. 123
    6.4.1 The benefits of using XFIC................................................................. 123
    6.4.2 The barriers in applying XFIC............................................................. 123
    6.4.3 How to choose the right frameworks............................................... 124
    6.4.4 How to start......................................................................................... 125
  6.5 FUTURE RESEARCH.................................................................................... 126

REFERENCES.................................................................................................... 127
LIST OF ACRONYMS

API       Application Programming Interface
ASC       Accredited Standards Committee
ASP       Application Services Provider
BFC       BizTalk Framework Compliant
BOM       Bills of Materials
B2B       Business-to-Business
B2C       Business-to-Consumer
CGI       Common Gateway Interface
CORBA     Common Object Request Broker Architecture
CPA       Collaboration Protocol Agreement
CPP       Collaboration Protocol Profile
CSS       Cascading Stylesheet Language
cXML      commerce XML
DCOM      Distributed Component Object Model
DOM       Document Object Model
DTD       Document Type Definition
EAI       Enterprise Application Integration
ebXML     Electronic business XML
eCo Framework Electronic Commerce Framework
e-business Electronic Business
e-commerce Electronic Commerce
EDI       Electronic Data Interchange
ERP       Enterprise Resource Planning
FTP       File Transfer Protocol
HTML      Hypertext Markup Language
HTTP      Hypertext Transport Protocol
HTTPs     Secure HTTP
ICE       Information and Content Exchange
IE        Microsoft Internet Explorer
IIOP      Internet Inter-ORB Protocol
IOTP      Internet Open Trade Protocol
JDBC      Java Database Connectivity
JMS       Java Message Services
JSP       Java Server Pages
JSWDK     Java Server Web Development Kit
J2EE      Java 2 Platform, Enterprise Edition
MIME      Multipurpose Internet Mail Extensions
OAG       Open Applications Group
OBI       Open Buying on the Internet
OFX Open Financial Exchange
OASIS Organization for the Advancement of Structured Information Standards
OOAD Object-Oriented Analysis and Design
ORB Object Request Broker
PIPs Partner Interface Processes
RDF Resource Description Framework
RMI Remote Method Invocation
RPC Remote Procedure Call
SAX Simple API for XML
SMEs Small and Middle Enterprises
SMTP Simple Mail Transfer Protocol
SOAP Simple Object Access Protocol
SOX Schema for Object-oriented XML
SQL Structured Query Language
SSL Secure Sockets Layer
TCP/IP Transmission Control Protocol/Internet Protocol
UDDI Universal Description, Discovery, and Integration
UML Unified Modeling Language
UN/EDIFACT United Nations / Electronic Data Interchange For Administration, Commerce and Transport
URI Universal Resource Identifier
URL Universal Resource Locator
VAN Value-Added Network
WF-XML Workflow XML
W3C World Wide Web Consortium
WSDL Web Services Description Language
xCBL XML Common Business Language
XDR XML Data Reduced
XFIC XML-based Frameworks for Internet Commerce
XLink XML Linking Language
XML Extensible Markup Language
XSL XML Stylesheet Language
XSLT XSL Transformations
1 Introduction

This chapter lays a background for the whole thesis. Section 1.2 explains the problems we target and Section 1.3 summarizes the approaches and contributions. To clarify the problems, Section 1.1 addresses some common concepts and related situations. Finally, Section 1.4 describes the organization of the thesis.

1.1 Situation

1.1.1 e-business and e-commerce

Electronic commerce (e-commerce) is the buying and selling of goods and services on the Internet. In practice, this term and “electronic business (e-business)” are often used interchangeably. Strictly, however, there are some differences, which can be seen from the definitions of Mesenbourg (1999), Bureau of the Census, the USA:

E-business is any process that a business organization conducts over a computer-mediated network. Business organizations include any for-profit, governmental, or nonprofit entity. Their processes include production-, customer-, and internal or management-focused business processes.

E-commerce is any transaction completed over a computer-mediated network that involves the transfer of ownership or rights to use goods or services. Transactions occur within selected e-business processes (e.g., selling process) and are “completed” when agreement is reached between the buyer and seller to transfer the ownership or rights to use goods or services.

The differences between e-business and e-commerce are that the latter emphasizes the transactions of transferring the ownership or rights to use goods or services, and the former includes e-commerce but also covers internal processes such as production, inventory management, product development, risk management, finance, knowledge management and human resources.

Generally there are two kinds of e-commerce: B2B (business-to-business) and B2C (business-to-consumer).

1 B2C is the retailing part of e-commerce on the Internet and B2B is the exchange of products, services, and information between businesses. Compared to B2C,

• B2B can be defined as a set of more formal business processes because more repetitive tasks take place between two businesses;
• B2B is the automation of server-to-server communication (one-way or two-way);

1 Besides B2C and B2B, there exist many other types of e-commerce with the X2Y patterns, e.g., C2C (consumer-to-consumer) and B2G (business-to-government).
• B2B needs some dynamic events to trigger suitable business processes.

1.1.2 e-procurement

Electronic procurement (e-procurement) is one of the fastest growing types of B2B e-commerce, generally defined as a business process to purchasing parts, components or services from business partners via the Internet. Zaharino (2000) forecasts the total dollar volume spent on procurement goods and services over the Internet to reach $520 billion in 2004. Service opportunities around the traditional set of consulting, deploying, and managing e-procurement systems are projected to grow from $1.4 billion in 1999 to $12.7 billion in 2004.

E-procurement is also one of the most fundamental types of B2B e-commerce for at least two reasons:

• It tends to be most labor-, paper- and time-consuming and requires much more repetitive work to be done after the procurement processes have been set up, thereby resulting in huge potential gains in automating it.
• It can easily be modeled as some formal business processes (cataloging, search, negotiation and payment), thus making it easier to model and to program.

Therefore, it is easy to understand that B2B e-procurement can be traced back to the middle of the 19th Century with the introduction of the telegraph (Gerhard 1999, p108-9). Computer-based e-procurement has undergone three development stages:

• EDI (Electronic Data Interchange), introduced in the late 1960s, is the exchange of routine business documents in a structured format between computer applications within and between companies. The goal is to eliminate the time and data entry associated with paper. Two specifications coexist, one is the United Nations’ UN/EDIFAC and the other is American ASC X12. The obvious disadvantage of EDI is the high cost because EDI is based on the value-added network (VAN). Small and middle enterprises (SMEs) are difficult to independently use it.

• OBI (Open Buying on the Internet), started in October 1996, is an HTTP-based framework for B2B e-commerce. Ubiquitous Internet is applied instead of VANs to reduce the cost and barrier of SMEs’ joining in B2B. But OBI still uses the rigid EDI formats as the transport media, thus limiting its extensibility to fit a wide variety of business requirements (OBI 1999). Moreover, EDI message formats are not easy to read although they are text-based.

• XML-based e-procurement, which started in February 1998 after XML (eXtensible Markup Language) specification 1.0 was published, is the solution based on the Internet and XML (see next Section). It is supposed in theory to fulfill the requirements of easy access via the Internet, and high customization via XML’s layer structure.

---

2 EDI message is a text-based format, with dictionaries of globally defined tag sets. An EDI message is layered, including such layers as Data Element, Compound Data Element, Data Segment, Loop, and Business Document. Thus it has some capability to represent tree-structured data. However, it is not as flexible as XML. This is because its number of layers is fixed. With XML, by contrast, you can define a data structure that has an arbitrary number of layers. (Maruyama et al., 1999, p 235-236).
extensibility. Ariba, Inc and Commerce One, Inc. are two representative companies offering XML-based e-procurement infrastructure and software.

1.1.3 XML

According to W3C (World Wide Web Consortium), XML is a universal format for structured documents and data on the Web. XML is a text format and makes use of tags (words bracketed by ‘<’ and ‘>’) and attributes (of the form name = “value”). Compared to HTML (Hypertext Markup Language), XML is extensible by allowing you to define your own tags as required. Figure 1 presents an example to show this characteristic. The first part is a simple HTML file that presents a three-row table. All tags (html, body, table, tr and td) are rigid and specified in HTTP (Hypertext Transfer Protocol) 1.0. The second part is an XML document to show the same table, and you can choose any tags (personnel, row and column) or delete any tags (table) as you like, but you have to add a header and to define DOCTYPE (personnel.dtd).

XML represents a family of technologies. XML 1.0 specification as the core defines what tags and attributes are. The latest version of XML 1.0 is the Second Edition published on Oct. 6, 2000. XML Schema 1 and 2 help developers to more rigorously and comprehensively define the structure, content and semantics of XML documents. XML Namespaces offers a solution to associate a URI (Universal Resource Identifier) reference with every single tag and attribute in an XML document allowing developers to use the same tag and attribute but different namespaces that enhance modularity of XML software. Besides, there is a growing set of optional modules that provide sets of tags & attributes, or guidelines for specific tasks:

- **Xlink** describes a standard way to add hyperlinks to an XML file.
- **Xpointer & Xfragments** are syntaxes for pointing to parts of an XML document. Xpointer is somewhat like a URL, but instead of pointing to documents on the Web, it points to pieces of data inside an XML file.
- **CSS**, cascading style sheet language, aims primarily at on-screen formatting, first developed for HTML but usable with XML without modification. Currently, Netscape and Microsoft Internet Explorer support it. It is popular to use XML to describe Web content and CSS for Web presentation in building Web sites (e.g., http://www.infoworld.com).
- **XSL**, XML stylesheet language, is the advanced language for expressing style sheets based on XSLT (XSL transformation) designed for rearranging, adding or deleting tags and attributes.
- **DOM**, document object model, is a standard set of function calls for manipulating XML files from a programming language.

All of the bold words above refer to http://www.w3.org/ where the latest progress and detailed information are available.

---

3 The DTD (Document Type Definition) file is to define the structure rules to write the XML document. An XML parser uses it to check that a tagged XML document conforms to the pre-defined document structure rules.
\begin{verbatim}
<html>
<body>
<table>
<tr> 
<td>Name</td> 
<td>Birth_date</td> 
<td>Sex</td> 
<td>Email</td> 
</tr> 
<tr> 
<td>Johan Ericsson</td> 
<td>19670707</td> 
<td>male</td> 
<td>joher@ida.liu.se</td> 
</tr> 
<tr> 
<td>Eva Svensson</td> 
<td>19701012</td> 
<td>female</td> 
<td>eva.svensson@ida.liu.se</td> 
</tr> 
</table>
</body>
</html>

<?xml version="1.0" ?>
<!DOCTYPE personnel SYSTEM personnel.dtd>
<personnel>
<row>
<column>Name</column> 
<column>Birth_date</column> 
<column>Sex</column> 
<column>Email</column> 
</row> 
<row> 
<column>Johan Ericsson</column> 
<column>19670707</column> 
<column>male</column> 
<column>joher@ida.liu.se</column> 
</row> 
<row> 
<column>Eva Svensson</column> 
<column>19701012</column> 
<column>female</column> 
<column>eva.svensson@ida.liu.se</column> 
</row> 
</personnel>
\end{verbatim}

\begin{figure}[h]
\begin{verbatim}
<html>
<body>
<table>
<tr> 
<td>Name</td> 
<td>Birth_date</td> 
<td>Sex</td> 
<td>Email</td> 
</tr> 
<tr> 
<td>Johan Ericsson</td> 
<td>19670707</td> 
<td>male</td> 
<td>joher@ida.liu.se</td> 
</tr> 
<tr> 
<td>Eva Svensson</td> 
<td>19701012</td> 
<td>female</td> 
<td>eva.svensson@ida.liu.se</td> 
</tr> 
</table>
</body>
</html>

<?xml version="1.0" ?>
<!DOCTYPE personnel SYSTEM personnel.dtd>
<personnel>
<row>
<column>Name</column> 
<column>Birth_date</column> 
<column>Sex</column> 
<column>Email</column> 
</row> 
<row> 
<column>Johan Ericsson</column> 
<column>19670707</column> 
<column>male</column> 
<column>joher@ida.liu.se</column> 
</row> 
<row> 
<column>Eva Svensson</column> 
<column>19701012</column> 
<column>female</column> 
<column>eva.svensson@ida.liu.se</column> 
</row> 
</personnel>
\end{verbatim}

\textbf{Figure 1} A comparison of HTML and XML using an example
1.2 Complication

XML promises to be a much better solution to realize both interoperability and customization in heterogeneous e-commerce environments. However, some problems are obvious to prevent wider adoption:

- **XML standards are under construction.** W3C is responsible for recommendations for XML standards. A specification in the stage of recommendation needs to undergo five stages: working draft, last call working draft, candidate recommendation, proposed recommendation, and recommendation (Jacobs, 2001). Most XML specifications have not reached the recommendation stage according to www.w3.org/TR/. One of significant specifications is the XML Schema, which is on the stage of proposed recommendation \(^4\) (010330). The year of 2000 witnessed two major changes for the XML Schema specification.

- **There are a large number of diverse ongoing frameworks, or industry standards, for Internet commerce.** e.g., eCo Frameworks, ebXML (e-business XML), UDDI (Universal Description, Discovery, and Integration), BizTalk, cXML (commerce XML), SOAP (Simple Object Access Protocol), WSDL (Web Services Description Language), xCBL (XML Common Business Library), and RDF (Resource Description Frameworks). Some frameworks are compatible with each other and some are not. Thus, they are creating a new issue of interoperability.

- **XML alone cannot be used in application development.** The XML document is static and stateless, it demands other languages to move it for session handling (e.g., request, response and process). Java could be a good choice to act on the role.

- **The training of XML and e-commerce is not easy.** For example, it is hard to design a project for training of e-commerce using XML. Such a project is subject to such constraints as cost, time and knowledge aggregation.

The first problem is hard to solve because time is necessary for the formal processes so as to guarantee the quality of XML standards. This thesis attempts to attack the last three problems.

1.3 Contributions

My approach is to:

- Describe the most important efforts according to a hierarchical model, a relationship model and a chronological model to help:
  - Understand what XML-based frameworks are.
  - Identify how they interact.

\(^4\) Proposed Recommendation means that the specification is table and that implementation experience has been gathered showing that each feature of the specification can be implemented. After review by the Consortium’s Advisory Committee, this specification will either be published as a recommendation, or republished as a candidate recommendation or as a working draft.
CHAPTER 1

- Implement an open-source demonstrator in order to:
  - Gain first hand experience of implementing B2B e-commerce using XML and Java.
  - Design a project for XML and e-commerce education.
  - Extract a feasible, though not optimal, development method.

The contributions of the thesis are summarized as follows:
- Survey the fifteen most important and active XML-based frameworks for Internet Commerce (XFIC) based on a large number of specifications. Part of results was presented in the International Conference on Enterprise Information Systems (Zhao and Sandahl 2000).
- Present three models to systematically understand 15 frameworks in depth for the requirements of e-commerce. A hierarchical model is provided to show the purpose and focus of XFIC. A relationship model is provided to show the cooperative and competitive relationships between XFIC. A chronological model is offered to look at XFIC developmentally.
- Provide guidelines for how to apply XFIC on e-commerce development.
- Implement a B2B e-procurement using Java and XML. This implementation demonstrates the feasibility of open-source or freeware and validates the complementary roles of XML and Java in e-commerce development.
- Implement auction-based dynamic pricing in the B2B e-procurement which led to some unexpected problems related to it.
- Identify a project for XML and e-commerce education. E-procurement is an appropriate option.

1.4 The organization of the thesis

The thesis is organized as follows:
- Chapter 1: Introduction.
- Chapters 2-4: XML-based Frameworks for Internet Commerce (XFIC):
  - 2: Presents a hierarchical model, a relationship model and a chronological model to understand XFIC.
  - 3: Focuses on three frameworks (ebXML, eCo Framework, and UDDI) to meet the big picture of e-commerce.
  - 4: Focuses on 12 frameworks. The frameworks (SOAP/BizTalk, cXML and ICE; OAG, RosettaNet, and Wf-XML, OFX; VoiceXML) are for XML messaging. The frameworks (WSDL and RDF) are for XML syntax and semantics. xCBL is for establishing basic XML vocabulary.
- Chapter 5: Documents an implementation of B2B e-Procurement.
- Chapter 6: Summary and Conclusions.
2 XML-based Frameworks for Internet Commerce (XFIC)

2.1 Introduction

2.1.1 Motivation
Currently, there are many XML-based frameworks or industrial initiatives emerging, for instance, BizTalk, RossetNet, cXML, xCBL, eCo Framework, Open Application Group, ebXML, UDDI, SOAP, ICE, RDF, OFX, and the like. They have some common characteristics:
- Each has its own specifications.
- Each is related to Internet commerce.
- Each is based on XML.
- Each addresses the issues of interoperability.
- Each is changing quickly.

Moreover, more and more new XML-based frameworks are emerging. This thesis aims to put these frameworks together and describe them in the context of Internet commerce, and address what they are and how they interact (see 2.2 & 2.3). Thus, a comprehensive comparison and contrast will be given in a hierarchical model, a relationship model and a chronological model. The objectives of the three models are to:
- Help systematically understand these frameworks.
- Save time from reading thousands-of-pages specifications.

In addition, we attempt to predict where these frameworks are headed in the future.

2.1.2 Selection criteria
If you read the survey of XFIC (Kotok 2000), you will find we could include more than 100 options. This thesis selects fifteen frameworks (see Table 1 & 2). The selection criteria are as follows:
- It must be XML-based with the aims of Internet commerce, B2B or B2C.
- It must be active, influential, and broadly known in industry. This can be judged from the frequency of reports in IT news, and the number of joint partners, and the clout of main players.
- It has some formal specifications published on its Website.
Table 1  Fifteen main XML-based frameworks addressed in the thesis

<table>
<thead>
<tr>
<th>Framework</th>
<th>Web site</th>
<th>Specifications</th>
<th>Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>ebXML (e-business XML)</td>
<td><a href="http://www.ebxml.org">www.ebxml.org</a></td>
<td>Technical Architecture 1.0.4</td>
<td>010216</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Business Process Specification Schema 0.90</td>
<td>010117</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Registry Service 0.83</td>
<td>010116</td>
</tr>
<tr>
<td>eCo Framework</td>
<td>eco.commerce.net</td>
<td>eCo Architecture for Electronic Commerce Interoperability</td>
<td>990629</td>
</tr>
<tr>
<td>UDDI (Universal Description, Discovery and Integration)</td>
<td><a href="http://www.uddi.org">www.uddi.org</a></td>
<td>Programmer’s API 1.0</td>
<td>000930</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Data structure Reference 1.0</td>
<td></td>
</tr>
<tr>
<td>BizTalk</td>
<td><a href="http://www.biztalk.org">www.biztalk.org</a></td>
<td>BizTalk Framework 2.0</td>
<td>Dec 2000</td>
</tr>
<tr>
<td>cXML (Commerce XML)</td>
<td><a href="http://www.cxml.org">www.cxml.org</a></td>
<td>CXML’s User’s Guide 1.1</td>
<td>Jun 2000</td>
</tr>
<tr>
<td>ICE (Information and Content Exchange)</td>
<td><a href="http://www.icestandard.org">www.icestandard.org</a></td>
<td>ICE Implementation Cook Book</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>ICE Protocol 1.1</td>
<td></td>
</tr>
<tr>
<td>OAG (Open Applications Group)</td>
<td><a href="http://www.openapplications.org">www.openapplications.org</a></td>
<td>OAG Interface Specification (OAGIS) 7.0.2</td>
<td>001124</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>OFX (Open Financial eXchange)</td>
<td><a href="http://www.ofx.org">www.ofx.org</a></td>
<td>OFX Specification 2.0</td>
<td>000630</td>
</tr>
<tr>
<td>RDF (Resource Description Framework)</td>
<td><a href="http://www.w3.org/RDF/">www.w3.org/RDF/</a></td>
<td>RDF Schema Specification 1.0</td>
<td>000327</td>
</tr>
<tr>
<td></td>
<td></td>
<td>RDF Model and Syntax Specification</td>
<td>990222</td>
</tr>
<tr>
<td>RosettaNet</td>
<td><a href="http://www.rosettanet.org">www.rosettanet.org</a></td>
<td>Business Dictionary</td>
<td>N/A</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Technical Dictionary Implementation Framework</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Partner Interface Processes (PIPs)</td>
<td></td>
</tr>
<tr>
<td>SOAP (Simple Object Access Protocol)</td>
<td><a href="http://www.w3.org/TR/2000">www.w3.org/TR/2000</a></td>
<td>SOAP 1.1</td>
<td>000508</td>
</tr>
<tr>
<td></td>
<td>/NOTE-SOAP-</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>20000508/</td>
<td></td>
<td></td>
</tr>
<tr>
<td>VoiceXML</td>
<td><a href="http://www.voicexml.org">www.voicexml.org</a></td>
<td>Voice XML 1.0</td>
<td>000505</td>
</tr>
<tr>
<td>WSDL (Web Services Description Language)</td>
<td>www-4.ibm.com/software/dev</td>
<td>WSDL 1.0</td>
<td>000925</td>
</tr>
<tr>
<td></td>
<td>eloper/library/w-wsdl.html</td>
<td></td>
<td></td>
</tr>
<tr>
<td>XCB-L (XML Common Business Library)</td>
<td><a href="http://www.xcbl.org">www.xcbl.org</a></td>
<td>XCB-L 3.0</td>
<td>001129</td>
</tr>
</tbody>
</table>
**Table 2  Fifteen frameworks and their short descriptions**

<table>
<thead>
<tr>
<th>Name</th>
<th>Short Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ebXML</td>
<td>Enables a global electronic marketplace where enterprises of any size and in any geographical location can meet and conduct business with each other through the exchange of XML-based messages. ebXML is a joint initiative of UN/CEFACT and OASIS.</td>
</tr>
<tr>
<td>eCo Framework</td>
<td>Offers a systematic way to view and organize e-commerce systems from 7 layers: network, market business service, interaction, document and information items. Main players are CommerceNet and Commerce One Inc.</td>
</tr>
<tr>
<td>UDDI</td>
<td>Is the building block that will enable businesses to quickly, easily and dynamically find and transact business with one another using their preferred applications. Main players are Microsoft, IBM and Ariba Inc.</td>
</tr>
<tr>
<td>BizTalk</td>
<td>Lets applications communicate reliably and securely via XML messaging. BizTalk server is one key to meet the goal. Main player is Microsoft.</td>
</tr>
<tr>
<td>cXML</td>
<td>Facilitates easy exchange of catalog content and transaction information in procurement between trading partners. The main player is Ariba Inc.</td>
</tr>
<tr>
<td>ICE</td>
<td>Facilitates the controlled exchange and management of electronic assets between networked partners and affiliates. Syndicated publishing is on focus. Main player is IDEAlliance, formed by GCA (Graphic Communications Association).</td>
</tr>
<tr>
<td>OAG</td>
<td>Offers massive XML formats and business integration scenarios to integrate enterprise systems. Main player is Open Applications Group that consist of vendors of enterprise systems.</td>
</tr>
<tr>
<td>OFX</td>
<td>Enables electronic exchange of financial data over the Internet. Main players are CheckFree, Intuit, and Microsoft.</td>
</tr>
<tr>
<td>RDF</td>
<td>Provides a data model and syntax that can be extended to address sophisticated ontology representation techniques. Main player is W3C.</td>
</tr>
<tr>
<td>RosettaNet</td>
<td>Defines the business processes in supply chain management and provides the technical specifications for data interchange. Main player is RosettaNet, a consortium including electronic components, IT and semiconductor companies.</td>
</tr>
<tr>
<td>SOAP</td>
<td>Is a lightweight protocol for exchanging XML documents in a decentralized and distributed environment. It has been submitted to W3C. Main players include Microsoft, IBM, and others.</td>
</tr>
<tr>
<td>VoiceXML</td>
<td>Defines dialogs between humans and machines in terms of audio files to be played, text to speech and speech recognition capabilities, and touch-tone input. Main players include AT&amp;T, IBM, Lucent, and Motorola.</td>
</tr>
<tr>
<td>WSDL</td>
<td>Is an XML format for describing network services as a set of endpoints operating on messages containing either document-oriented or procedure-oriented information. IBM and Microsoft are the main players.</td>
</tr>
<tr>
<td>Wf-XML</td>
<td>Uses XML to model data transfer requirements between workflows to achieve workflow interoperability. Main player is MFMC (Workflow Management Coalition).</td>
</tr>
<tr>
<td>xCBL</td>
<td>Offers basic vocabulary definitions in e-commerce. Main player is Commerce One, Inc.</td>
</tr>
</tbody>
</table>
2.1.3 Related work

The research on XML-based frameworks is extensive in industry. The main references come from the specifications produced by big companies or non-profit consortiums. Table 1 shows 15 frameworks and their corresponding specifications, which this thesis is mainly based on. We can receive their development information by reading their press releases published on their Web sites.


Two papers discuss several frameworks together. Zhao and Sandahl (2000) make a survey of the architecture and message definition of BizTalk, cXML, eCo Framework, ICE, IOTP (Internet Open Trade Protocol), Open Applications Group, RosettaNet, and xCBL. It also analyzes their purposes, relationships (competitive and cooperative), and how to use the frameworks in developing e-commerce systems. But it does not address ebXML (no specifications were published at the date of publication), Wf-XML, RDF, VoiceXML, and so on.

Shim et al (2000) collect five important B2B e-commerce frameworks (i.e., OBI, eCo Framework, RosettaNet, cXML, and BizTalk), and compare the frameworks on aspects of industry target, architecture, security, communication protocol, message format, ontology, and catalog). But it is not about XML-based frameworks and most of them are outdated, for example, BizTalk is based on BizTalk 1.0, not SOAP-based BizTalk 2.0 (Microsoft 2000).
2.2 From XML to Xfic

2.2.1 XML Applications

XML can be applied in a wide range of areas. Maruyama et al. (1999) concluded three kinds of applications:

1. Use XML to describe metacontent regarding documents or on-line resources.
2. Use XML to publish and exchange database contents.
3. Use XML as a messaging format for communication between application programs.

Actually, it can be used in much broader areas. For instance, an innovative use of voice, VoiceXML (see 4.5), was developed in 2000 to support human-computer dialogs via spoken input and audio output (Danielsen 2000). However, if we consider a database as a special kind of application, XML has two kinds of applications:

1. Describe metacontent of any structured data using extensible tags so that people and machine can understand. Here any structured data includes spreadsheets, configuration parameters, financial transactions, technical drawings, etc.
2. Represent a transport media so that any heterogeneous computer system can communicate. XML is totally in text format (of course constrained), so it can be easily read and processed by any systems at least in theory.

2.2.2 The roles of XML in Internet commerce: interoperability

Interoperability means the ability of separate systems to be linked together and then operate as if they were a single entity. One critical requirement for Internet commerce is interoperability because business transactions and communications have been conducted in highly heterogeneous environments with respect to aspects of platform, language, protocol, server, and vendor. XML as text format theoretically eases the problem. Whatever environments Application X resides in, it can send to Application Y an XML file that represents its own request. And whatever environments Y resides in, Y can read the XML file and process it and then send another XML document as a response to X.

Concretely speaking, XML has played two roles in Internet commerce corresponding to two kinds of XML applications mentioned above:

1. Describe the content of Web applications and services. Figure 2 shows the StockQuote service defined by WSDL (see 4.6.1). The service supports a single operation called getLastTradePrice. The request takes ticker symbol of type string, and returns the price as a float, which is defined inside types. Inside element types is the XML schema definition.
2. Represent transport data between applications. Figure 3, for example, show the SOAP (see 4.1.1) request and response. It requests the last trade price of stock ERICY, and produces a response of price 20.5.
<?xml version = "1.0"?>
<definitions name = "StockQuote"
    targetNamespace = "http://example.com/stockquote/definitions"
    xmlns:tns = "http://example.com/stockquote/definitions"
    xmlns:xsd1 = "http://example.com/stockquote/schemas"
    xmlns:soap = "http://schemas.xmlsoap.org/wsdl/soap/"
    xmlns = "http://schemas.xmlsoap.org/wsdl/">
    <types>
        <schema targetNamespace="http://example.com/stockquote.xsd"
            xmlns="http://www.w3.org/1999/XMLSchema">
            <element name="LastTradePriceRequest">
                <complexType>
                    <all>
                        <element name="tickerSymbol" type="string"/>
                    </all>
                </complexType>
            </element>
            <element name="LastTradePriceResult">
                <complexType>
                    <all>
                        <element name="price" type="float"/>
                    </all>
                </complexType>
            </element>
        </schema>
    </types>
    <message name = "GetLastTradePriceRequest">
        <part name = "body" element = "xsd1:LastTradePriceRequest"/>
    </message>
    <message name = "GetLastTradePriceResponse">
        <part name = "body" element = "xsd1:LastTradePriceResult"/>
    </message>
    <portType name = "StockQuotePortType">
        <operation name = "GetLastTradePrice">
            <input message = "tns:GetLastTradePriceRequest"/>
            <output message = "tns:GetLastTradePriceResponse"/>
        </operation>
    </portType>
    <binding name="StockQuoteSoapBinding" type="tns:StockQuotePortType">
        <soap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http"/>
        <operation name="GetLastTradePrice">
            <soap:operation soapAction="http://example.com/GetLastTradePrice"/>
            <input>
                <soap:body use="literal" namespace="http://example.com/stockquote.xsd" encodingStyle="http://schemas.xmlsoap.org/soap/encoding"/>
            </input>
            <output>
                <soap:body use="literal" namespace="http://example.com/stockquote.xsd" encodingStyle="http://schemas.xmlsoap.org/soap/encoding"/>
            </output>
        </operation>
    </binding>
    <service name="StockQuoteService">
        <documentation>My first service</documentation>
        <port name="StockQuotePort" binding="tns:StockQuoteBinding">
            <soap:address location="http://example.com/stockquote"/>
        </port>
    </service>
</definitions>

Figure 2   A Service Definition of StockQuote using WSDL (WSDL 2000)
SOAP requests the last trade price of **ERICY:**

```xml
<SOAP-ENV:Envelope
    xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
    SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
    <SOAP-ENV:Body>
        <m:GetLastTradePrice
            xmlns:m="http://example.com/stockquote/definitions">
            <tickerSymbol>ERICY</tickerSymbol>
        </m:GetLastTradePrice>
    </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

SOAP gives a response of **20.5:**

```xml
<SOAP-ENV:Envelope
    xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
    SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
    <SOAP-ENV:Body>
        <m:GetLastTradePriceResponse
            xmlns:m="http://example.com/stockquote/definitions">
            <price>20.5</price>
        </m:GetLastTradePriceResponse>
    </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

**Figure 3**  Request stock price last trade and response using SOAP
### 2.2.3 A generic model for XFIC

The goal of XML-based frameworks is to achieve application interoperability via XML messaging, or exchanging XML documents between applications. It provides an open template that allows as many entities as possible to realize their messaging. Figure 4 demonstrates a generic model for XFIC. In this model, we find that XFIC should solve at least two problems:

- Define an XML format that the applications involved can accept and understand. The XML format is always defined by an XML DTD and/or schema.
- Design a mechanism for the applications to send and receive XML documents securely and reliably. A general solution is to bind suitable transport methods with the XML format.

Note the conditions that the generic model is applied to. They are extremely important in differentiating such models as client/server. These conditions include:

- Two applications are **open or universally accessible** via standard-based Web services. The services may be registered in a marketplace, by registry service providers or owned by themselves.
- Two applications are **loosely coupled** components. Their interactions may be performed once or frequently. Here ‘loosely-coupled’ is against those tightly coupled infrastructures such as CORBA (Common Object Request Broker Architecture), DCOM (Distributed Component Object Model), and RMI (Remote Method Invocation). For two tightly coupled applications, if one is changed, the other is very likely to break.
- Two applications are on **equal position**, and neither can control the other and neither just serve for the other.
- Two applications **share some design goals and specifications** so as to understand the XML document transported between them.

These requirements in Internet commerce are fundamental. In particular, in B2B e-commerce, to exchange business information to realize the most ordinary scenarios such as transactions or information query, two applications belonging to two organizations are independent but similar enough to allow them to communicate.
2.2.4 Philosophy behind XFIC

A framework is a generic template that provides a desired functionality (Shim et al., 2000). The philosophy behind XML-based frameworks comes from the apparent conflict between extensibility and interoperability of XML. XML can be extensible by defining any tags, any attributes and any structures that you expect, and this is very useful in fulfilling your customized business requirements. However, when each organization attempts to create its proprietary framework of XML documents, a new interoperability issue arises, i.e., how to interoperate these customized frameworks.

The essence of XML-based frameworks is a compromise between extensibility and interoperability to maximally utilize XML. To achieve this goal, these frameworks have normally conducted the following work:

- XML messaging is clearly modeled, e.g., one way or request-response.
- Message structure is clearly defined by XML DTD, schema and namespace. Generally, a message consists of a header and a body. The header comprises of overall information about this message such as sender, receiver and transport mechanism whereas the body includes layered business transaction data.
- Some tags and attributes are proposed. For instance, the tag of product price, a framework may recommend you to use <Price> instead of <price> or <PRICE>.
- XML transport is bound with open Internet standards such as HTTP (Hypertext Transfer Protocol) and SMTP (Simple Mail Transfer Protocol).
- The method of extension and evolution is enabled.
2.2.5 XFIC and Web services

The notion of Web services exploded in June 2000 when Microsoft announced its .NET strategy for Internet-based applications (Sullivan 2001). W3C organized a workshop on Web services in April 2001 (w3.org/2001/03/WSWS-popA/), where different companies offer different explanations about the concept. Considering its post paper and the reports on hand, I summarize the characteristics of Web services as follows:

- Interoperable via standards: Web services can communicate one another no matter what platform, language and vendor they depend on. The standards involve XML, SOAP and WSDL (Sullivan 2001; Mendelsohn 2001; Kirtland 2001).
- Loosely coupled: the interaction between Web services is short-time partnership, rather than systematic in the client/server architecture (Ferguson 2001).
- Web services represent business functionality via standard interfaces (Sullivan 2001).
- Web services can be registered and discoverable via open Registry (Ferguson 2001).

Figure 5 shows the relationship between XFIC and Web services development: XFIC, together with XML-related protocols by W3C, are becoming the cornerstone of Web services development. With the popularity of Web services, XFIC will become more and more important.

XFIC-based infrastructures, or frameworks called by industry, are emerging through providing development environments and tools to ease Web services development. The noticeable infrastructures include Microsoft’s .NET (Parthasarathy 2000), IBM’s Application Development for e-Business (IBM 2000), Sun’s SUN ONE (Open Net Environment) (Sun 2001), and Oracle’s Oracle9i Dynamic Services (Oracle 2000). More infrastructures will certainly emerge to compete on market. These infrastructures are to offer means of building, aggregating, deploying and managing Web services.

The position of XFIC lies between Infrastructure layer and Application layer. XFIC offer the constraints to limit XML extensibility to achieve higher interoperability. And XFIC are established on the ground of TCP/IP protocol suite, which is the typical four-layer model (Stevens 1994). HTTP and SMTP are indispensable for XFIC.

The black box of XFIC will be further analyzed in subsequent chapters (Section 2.3; Chapter 3 & 4).

---

5 One of definition: Web services are software components that represent business functionality that can be accessed by users via applications or other Web services using standard protocols (Sullivan 2001).
Interoperable via standards
Loosely coupled
Represent business functionality with standard interfaces
Registered and discoverable
Vendor-related
Development environments and tools
Based on XFIC
UDDI, SOAP, WSDL, ebXML, etc
To become standards
Based on XML protocols by W3C
Bound to HTTP and SMTP
HTTP, SMTP, FTP, etc
TCP, UDP
IP

Figure 5  XFIC and Web services development
2.3 Three models for describing XFIC

2.3.1 Business processes for Internet commerce

XFIC are proposed to fulfill the requirements of Internet commerce. But what are the requirements we are targeting? It is a hard question, but we cannot evade them as we address XFIC. This section will simply analyze the basic business processes in Internet commerce, and decompose Internet commerce into several united subsystems without overlaps, conflicts and omitted functions. Here “basic” is viewed from our current knowledge and a current technology standpoint.

Internet commerce involves the activities of buying and selling over the Internet. Suppose that Company A wants to do business with some business partners. Company A can use the Internet to search for such partners and then negotiate and sign a contract using email.

Now Company A requires a more efficient way to automate the processes to conduct B2B Internet commerce. Then Company A will interact with the applications of some business partners. Such applications are open on the Internet, known as Web services designed to offer the operations of buying and selling. The company needs to perform the following processes to find the right Web services (see Figure 6):

- Register: it registers its own services on the Internet like its business partners.
- Search: it uses suitable search engines to find all possible Web services.
- Negotiate: it negotiates with these services.
- Contract: it signs contracts with all appropriate Web services.
- Configure: it configures its own services to enable direct interactions via the Internet.

Afterwards Company A will conduct direct transactions with the services of its business partners. Suppose that one of partners is Company B. It needs to do the following processes (Figure 6):

- Send: it sends an XML document.
- Receive: it receives the XML document and validates it.
- Understand: it knows the meaning of the XML document.
- Process: it processes the XML document according to its content, and produces another XML document for sending.

Figure 6 shows the structure of the business processes as layered, which results in our subsequent hierarchical model.
The requirements for business processes in Internet commerce

Figure 6  The requirements for business processes in Internet commerce
2.3.2 A Hierarchical Model

This section accounts for how the fifteen XML-based frameworks fit the requirements of Internet commerce discussed in the last section. Our goal is to describe how interoperation using XML-based frameworks can be achieved.

Figure 7 illustrates the research approach. We start with the generic model for XFIC (see Figure 4), i.e. XML messaging between two applications for e-business. As we go up, we aim to look at the big pictures of e-commerce. In this case, we assume that the applications are not directly specified and you need to find them through a registry-and-search mechanism. As we go down, we are to focus on messaging between applications in detail. Here the applications are known to each other and you need to know how to message for easy implementation, e.g., message document and transport, semantics & syntax, and vocabulary.

![Diagram](Image)

**Figure 7** Go up and down from the generic model for XFIC
2.3.2.1 The big picture as the top layer

We suppose that Company A requires some frameworks to design its applications or Web services so as to fulfill the process requirements of Register, Search, Negotiate, Contract and Configure in Internet commerce.

Three frameworks can meet some requirements to some degree, ebXML, eCo Framework and UDDI. The match is shown in the top layer of Figure 8. We call this layer the big picture, meaning that we are looking for the big picture of Internet commerce. Chapter 3 discusses the three frameworks one by one.

- ebXML: it can be used in Register, Search, and Configure but not Negotiate. Its specifications are still under construction.
- eCo Framework, used in Register and Search, but not Negotiate and Configure. It has wrapped up since it published its specifications in 1999.
- UDDI, used in Register, Search and Configure but not Negotiate. It works well, but you may be subject to the constraints of three main players (Microsoft, IBM and Ariba).

This layer looks simple; in fact, the automation processes face various barriers. The reasons stem from technology, legal and social issues.

2.3.2.2 Messaging, Syntax & Semantics, and Vocabulary

Suppose that two Web services owned by Company A and Company B are specified and known. They need to choose frameworks to conduct transactions between specified services. The business processes include Send, Receive, Understand, and Process.

- Send and Receive can be merged and replaced by Transport in technical language. What they send and receive requires an agreement about the message structure.
- Understand requires the services to understand tokens (vocabulary) and meanings (syntax and semantics).
- Process represents the business logic of Web services but produces the right XML document for sending. Thus it needs a formal XML message structure, which is used as transport media.

Therefore, we translate the requirements of business processes into technical requirements: transport, message structure, syntax & semantics, and vocabulary. Because message structure and transport mechanisms are tightly connected in an implementation, we put them together into messaging. Figure 8 displays messaging, syntax & semantics, and vocabulary in the hierarchical model.

Messaging

Messaging offers transport mechanisms and defines message structure. SOAP delivers a general solution using RPC (Remote Procedure Call). Its encoding of normal “element” in Document Objet Model and simple representation of remote method call allow it to simply and efficiently exchange data in the environment of Web services.
Its transport can be bound to HTTP and SMTP/MIME. Its message structure (see Figure 2) is as follows:

\[
\text{[envelope + [header + body]} ],
\]

where
- envelope includes transport protocol information;
- header includes the general information about the body;
- body represents the payload or transaction data.

BizTalk is an application of SOAP\(^6\), and it adds more constraints but lets the services achieve more reliability and security during the processes of messaging. The solution is to offer a BizTalk Framework Compliant (BFC) server, which is responsible for sending and receiving XML documents.

Some special frameworks are also proposed for the special applications. We categorize them into four types according to their application characteristics.

1. **Specific business operations** handle business functions. cXML aims at e-procurement and marketplace. ICE is for 1-to-n syndicated subscription and publication, which can be used in marketing, sales and so on.

2. **Enterprise application integration** (EAI) handles the integration of business processes. OAG targets the integration between components produced by the vendors of enterprise resource applications (ERP) and supply chain management. RosettaNet identifies some general patterns representing the interaction between supply chain partners and gives its solutions to each pattern. Both OAG and RosettaNet offer scenario-based integration methods, thereby making development rapid but the implementation is subject to the local conditions. Wf-XML designs a general way to integrate workflow management systems using XML messaging.

3. **Vertical industry** handles industrial application. OFX is for exchanging financial data between customers and financial institutions. Many similar frameworks coexist.

4. **Specific applications** handle such applications as VoiceXML that supports human-computer dialogs via spoken input and audio output.

**Syntax and Semantics**

Syntax and semantics are to make XML document understandable especially for computers. For example, in SOAP’s body elements, you can theoretically write any elements and attributes. This would, however, certainly imply many difficulties for

\(^6\) SOAP might be replaced by XMLP (XML Protocol), the ongoing W3C project (w3.org/2000/xp). XMLP enables pieces of software sitting on one side of the Web—the clients of a service—-to remotely execute functionality offered by pieces of software at the other end of the Web—the clients of a service—the providers of the service. XMLP shares a number of characteristics with prior art systems such as RPCs, Microsoft DCOM, OMG’s CORBA or Java RMI. It is unique in its ability to make use of, and integrate with other Web technologies (Moreau and Fablet 2001). But there is no formal specifications of XMLP published (010417).
receiving applications in understanding and processing SOAP documents. Syntax and semantics account for problems by defining the grammar and metadata for common uses.

- RDF provides a general approach to describing metadata of any resource. A service understands the meaning of XML via the metadata.
- WSDL is to describe Web services, telling the visitors the name of the services, detailed operations and related configuration information.

**Vocabulary**

Vocabulary is to formalize the use of tags in XML. Human beings create plenty of words that create problems for computers. For example, if we create a tag to use “bike”, “Bike”, “BIKE”, “bicycle”, “Bicycle” or “BICYCLE”, we will face difficulties.

xCBL is to provide the fundamental vocabulary in e-commerce so that other frameworks do not need to create vocabulary from scratch. Such fundamental words include company, service, product, data and time, location, catalog, purchase order, and invoice.

To create a complete dictionary, we require all industries to contribute to it. XML.org is set as a repository to store the words.

**2.3.2.3 Main function and subsidiary functions**

We put one framework into just one layer in the hierarchical model (Figure 8). This matching is based on the main function of each framework. In fact, each framework involves many functions. Table 3 shows their functions involved. Bold X reflects the main functions and normal x represents the subsidiary functions. We find:

- ebXML is intended to offer nearly all functions that e-business requires.
- Most frameworks offer functions of syntax and semantics, because defining message structure certainly includes some syntax and semantics issues. Obvious conflicts exist in framework establishment.
Figure 8  A hierarchical model for XFIC
Table 3  XML-based frameworks and their functionality

<table>
<thead>
<tr>
<th>Name</th>
<th>Reg</th>
<th>Search</th>
<th>Contr</th>
<th>Config</th>
<th>Msg</th>
<th>S&amp;S</th>
<th>Voc</th>
</tr>
</thead>
<tbody>
<tr>
<td>ebXML</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>eCo Framework</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>UDDI</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SOAP</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>BizTalk</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>cXML</td>
<td>X</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ICE</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>OAG</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RosettaNet</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>OFX</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>VoiceXML</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RDF</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>WSDL</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Wf-XML</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>xCBL</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
</tbody>
</table>

Reg---Registry  
Contr---Contract  
Config---Configuration  
Msg---Messaging  
S&S---Syntax and Semantics  
Voc---Vocabulary  

X---major concentration  
X---involved
2.3.3 A Relationship Model

The relationship model describes competitive and cooperative relationship of the 15 frameworks. An ideal solution is to offer one framework like ebXML to meet all business requirements in e-commerce. But it is not the case in reality because it involves massive factors around interoperability and customization. A feasible solution is to choose the most consistent portfolio composed of different frameworks in different layers. One of important judgments of consistent is their relationship of cooperation and competition.

Figure 9 shows their relationships. In the middle circle, three leading groups deliver a top-level competition.

- eCo Framework is the earliest framework to address such requirements as registry and search. But it wrapped up in 1999 (see 2.3.4 and 3.2).
- The ebXML project started in 1999 but developed very quickly. Till now, however, it has not offered a suit of complete and proven specifications.
- The package of UDDI/WSDL/SOAP works together and grows very quickly than. Considering the clout of Microsoft and IBM, they are gaining more attention.

ebXML and UDDI/WSDL/SOAP are competitive although they are collaborative to some extent. Outside the middle circle, nine frameworks are looking for their special niches. OAG and RosettaNet aim at the issues of enterprise application integration (EAI) and have some common goals, thus they may cooperate together. Recent development envisages both actively support ebXML.

Wf-XML and cXML support the group of UDDI. Wf-XML expressly supports the BizTalk Framework. Ariba’s cXML should support its directly participating UDDI activities, but from its latest specification (Ariba 2000), we have not found how it supports UDDI.

xCBL support three groups. Some part of xCBL focuses on e-procurement, and thus xCBL is competitive with cXML. The competition between xCBL and cXML represents that of two leading e-commerce software companies: Commerce One and Ariba.

We have not observed any obvious examples of RDF, ICE, OFX and Voice supporting any group.

Among the cooperative and competitive relationships, Microsoft and IBM play the leading roles in UDDI/WSDL/SOAP and even ebXML, and Sun Microsystems, HP, and Oracle are normally opponents. However, more cooperation obviously arises than competition during producing these frameworks, because they envisage that without open, shared standards, costs grow faster than value, and value rises faster than cost only when participants share open standards (IBM 2000).
The relationships between 15 frameworks

**Figure 9** The relationships between 15 frameworks
2.3.4 A Chronological Model

XFIC have been quickly developing since XML specification 1.0 was published in Feb 1998. The chronological model describes 15 frameworks from their evolutions. Figure 10 briefly shows the development of 15 frameworks. Of these frameworks, only the project of eCo Framework, initiated as early as June 1998, wrapped up in October 1999. Others are still working cooperatively and competitively.

September 2000 is a significant point to separate the development of XFIC into two stages. The first stage tried to define XML formats and XML messaging to integrate applications. The characteristic is proprietary:

- Proprietary XML formats.
- Proprietary approaches of XML messaging.
- The presence of applications is proprietary without commonly accepted interfaces.

The second stage took place after September 2000 because of the emergence of UDDI and WSDL as well as the broad attention of Web services.

- Proprietary applications can be described as open Web services with common interfaces using WSDL.
- SOAP is becoming more and more popular in XML messaging. SOAP was originally proposed by Microsoft, IBM agreed to support it in April 2000, and ebXML also decided to support it in February 2001.
- The combination of UDDI, WSDL, and SOAP sets a foundation to develop interoperable Web services and makes developers envisage a clear vision of developing e-commerce.
- Meanwhile big companies are producing development environments and tools to ease Web services development, e.g., Microsoft’s .NET (Parthasarathy 2000), IBM’s Application Development for e-Business (IBM 2000), Sun’s SUN ONE (Open Net Environment) (Sun 2001), and Oracle’s Oracle9i Dynamic Services (Oracle 2000).

The trends of XFIC can be foreseen as follows:

- XFIC are the cornerstone to develop interoperable Web services. The development frameworks and tools that support XFIC ease the Web services development. We will find more cooperation on XFIC and more competition on the implementation of XFIC.
- ebXML can enrich the development of XML-based Web services and make the technology more open.
- The frameworks provided earlier than September 2000, especially cXML, ICE, RosettaNet, WF-XML, OFX and RDF, have to evolve their frameworks to adapt to Web services development. More cooperative relationships between these frameworks are expected to occur.
- Building the XML vocabulary, basic and industrial, is becoming more and more urgent. xCBL could be adopted under some conditions.
1: 4/00 (April 2000), IBM decided to take SOAP.
2: 6/00, BizTalk Framework 2.0 uses SOAP.
3: 2/01, ebXML takes SOAP.
Note: XML Specification 1.0 was officially recommended by W3C in 2/98.

**Figure 10** The development of 15 frameworks
Chapter 2 provides a hierarchical model to put XFIC together. Going up from the generic model (Figure 4), we assume that the applications are not directly specified and they need to find each other through a registry and search mechanism. There are three frameworks mentioned in the layer: ebXML, eCo Framework, and UDDI. This chapter will investigate them one by one and compare them on aspects of objective, architecture, registry, marketing strategy, disadvantage and maturity.

3.1 ebXML

ebXML was initiated by the United Nations Center for Trade Facilitation and Electronic Business (UN/CEFACT) and the Organization for the Advancement of Structured Information Standards (OASIS). They have joined forces to standardize XML business specifications internationally and to develop a technical framework that will enable XML to be utilized in a consistent manner for the exchange of all electronic business data. The vision of ebXML is ambitious, i.e., *Creating A Single Global Electronic Market*.

The ebXML framework provides a possibility to

- Preserve EDI’s substantial investments in business processes
- Exploit XML’s technical capabilities

3.1.1 System overview

Figure 11 show a high-level interaction between Company A and B to conduct e-business. It begins with Company A requesting ebXML specifications and related scenarios to build its ebXML-compliant systems. Then it can register its implementation details in ebXML registry systems. Thus Company B, who does not know Company A before, can query the company profiles and find that Company A is a suitable business partner. Company B sends a request to Company A stating that they would like to engage in a business scenario using ebXML. Company B acquires an ebXML compliant shrink-wrapped application (ebXML TAPT, 2001).

Before engaging in the scenario Company B submits a proposed business arrangement directly to Company A’s ebXML compliant software interface. The proposed business arrangement outlines the mutually agreed upon business scenarios and specific agreements on how it wants to conduct business transactions with Company A. The business arrangement also contains information pertaining to the messaging requirements for transactions to take place, contingency plans, and security-related requirements. Company A and B are now ready to engage in e-business using ebXML (ebXML TAPT, 2001).
Figure 11  A high level overview of the interactions using ebXML (ebXML TATP, 2001)
3.1.2 Business Operational View and Functional Service View

One of the important characteristics of ebXML is to distinguish business and IT activities to enhance the communication between business and IT people. ebXML sees business transactions from two views, Business Operational View (BOV) and Functional Service View (FSV), controlled by BOV standards and FSV standards respectively (ebXML TAPT, 2001).

BOV, not mandatory in ebXML, addresses the business process issues:
- The semantics of business data in transactions and associated data interchanges.
- The architecture for business transactions, including operational conventions, agreements and arrangements and mutual obligations and requirements.

FSV addresses the supporting services meeting the mechanistic needs of ebXML. It serves as a reference model that may be used by commercial software vendors to help guide them during the development process. It focuses on the information technology aspects of:
- Functional capabilities;
- Service interfaces

3.1.3 Registry and Repository

Registry and Repository are the key to enable information sharing between interested parties. Registry is a mechanism to allow process owners to submit, classify, register and update mapping templates, business process specifications, and data interchange specification. A Repository is a location or a set of distributed locations where documents pointed at by the Registry reside and from which they can be retrieved by conventional means such as HTTP and FTP (File Transfer Protocol).

Theoretically, such a mechanism is simple:
- The Registry clients register some objects with the Registry system;
- The Registry system assigns Unique Identifiers (UIDs) to all objects within the system;
- The system stores the objects with their UIDs in Repository;
- The Registry clients update, delete, query and retrieve the objects of interest.

However, implementation is complex, especially to meet the requirements of application and Registry. ebXML is conducting the following activities to ease the problem (ebXML RPT, 2001):
- Define Registry Services and related Registry Interfaces. A set of Registry Services are provided access to registry content to the Registry clients. These services allow Registry and its clients to conduct request and response via exposed interfaces. The interfaces include two types. One is exposed by the Registry and the other is by the Registry clients. For example, the interface of RegistryService is the principle
interface implemented by the Registry. A Registry client can get this interface from a connection to a registry. RegistryService has two methods:

- **getObjectNameManager()**: to return the ObjectManager interface implemented by the registry service. ObjectManager is to implement object life-cycle management functionality of the Registry.

- **getObjectQueryManager()**: to return the ObjectQueryManager interface implemented by the registry service. ObjectQueryManager is to implement the ObjectQuery management service of the Registry.

- Design Registry Information Model, the metamodel for the Registry, to define what types of objects are stored and how stored objects are organized in the Registry.

- Collaboration Protocol Profile (CPP), a document to allow a trading partner to express their supported business processes and service interfaces so that trading partners can be universally understood by other ebXML-compliant trading partners. More information is provided in the next section.

- Define the types of documents (DTDs) that move between the Registry and Registry clients.

### 3.1.4 CPP and Collaboration Protocol Agreement

CPP describes the specific capabilities that a trading partner supports as well as the service interface requirements that need to be met in order to exchange business documents between the trading partners. Each trading partner may register one or more CPP documents with an ebXML-compliant Registry system. CPP contains essential information about the trading partner including contact information, industry classification, supported business processes, interface requirements and messaging service requirements. CPP may also contain security and other implementation-specific details. Each ebXML-compliant trading partner should register its CPP in an ebXML-compliant Registry system, thus providing a discovery mechanism that allows trading partners to

- Find one another;
- Discover the business process that other trading partners support.

A Collaboration Protocol Agreement (CPA) is a document that represents the intersection of two CPPs and is mutually agreed upon by both trading partners who wish to conduct e-business using ebXML. A CPA describes:

- The messaging service;
- The business process requirements that are agreed upon by two or more trading partners.

Conceptually ebXML supports a three-level view of narrowing subsets from CPP to CPA for e-business (see Figure 12). The outermost scope relates to all of the capabilities that a trading partner can and will support, and the middle scope represents what it can support, and the inner is the transactions it agrees to work with.
Three-level view of Collaboration Protocol Agreement (ebXML TATP, 2001)

Figure 12 Three-level view of Collaboration Protocol Agreement (ebXML TATP, 2001)

### 3.1.5 Modeling business interactions

ebXML views e-business as multiple party collaboration. To enable business partners to collaborate, the ebXML Business Process Specification Schema offers an approach to model the interaction between them (ebXML CMG, 2001). The overall architecture is modeled as five chunks:

- **Business Collaboration**, a set of interactions between business partners. Each partner plays one or more roles in the collaboration. Binary Collaborations are between two roles only and expressed as a set of `BusinessActivities` between two roles. The `BusinessActivities` can be ‘atomic’, i.e. the activity of conducting an atomic `BusinessTransaction`, or ‘composite’, i.e. the activity of conducting another Binary Collaboration. Binary Collaboration can be synthesized into multi-party collaborations. Figure 13 illustrates the relationship between `BinaryCollaboration` and `MultiPartyCollaboration`.

- **Business Transaction**, an atomic unit of work in a business collaboration. A business transaction is always conducted between two business partners as two opposite roles (requester and responder). Figure 14 shows `AuthorizingRole` as an important class in `BusinessTransaction` linking `BinaryCollaboration` and `BusinessActivity`.

- **Business Message**, a set of business documents and signals exchanged between the requesting and responding roles to realize business transactions. The UML class diagram refers to Figure 15.

- **Document Definition**, the basic building block for information structure. The UML class diagram refers to Figure 15.

- **Choreography**, to combine and coordinate Business Transaction, Business Collaboration, Message Exchange and Document Definition in terms of States (i.e.
start state, completion state, activity state and synchronization state). Its UML class
diagram refers to Figure 16.

EbXML uses both UML and DTD to model the interactions between business partners. Each element found in the DTD refers to the corresponding class in the UML class
diagram. Element binary-collaboration, representing the class of BinaryCollaboration in
Figure 13, is defined by DTD as shown in Figure 17.

The conversion between the UML and DTD version of modeling business interactions
is based on production rules. The rules provide the prescriptive definition necessary to
translate a UML class specification into an XML document and the well-formed rules
(ebXML CMG, 2001). One of translation between the DTD and UML is name
conversion, e.g., \textit{binary-collaboration} and \textit{BinaryCollaboration}, \textit{business-activity} and
\textit{BusinessTransaction}.

\subsubsection{3.1.6 Messaging Service}

The ebXML Messaging Service must provide a secure, consistent and reliable
mechanism to exchange ebXML messages between users of the ebXML infrastructure
over various transport protocols, shown in Figure 18. The service prescribes formats for
all messages between distributed ebXML components including Registry mechanisms
and compliant user applications. It must also utilize and enforce the “rules of
engagement” defined in a CPA (ebXML TAPT, 2001). It also provides an abstract
interface whose functions include:

\begin{itemize}
\item Send, to send an ebXML message.
\item Receive, to receive an ebXML message.
\item Notify, to provide notification of unexpected events.
\item Inquire, to provide a method of querying the status of the particular ebXML
  message interchange.
\end{itemize}

Figure 19 illustrates the logical structure of an ebXML message. The message may
consist of an optional transport protocol-specific outer communication protocol
envelope (e.g., HTTP and SMTP) and a protocol-independent ebXML message
envelope. The ebXML message envelope may be packaged using the MIME
(multipurpose Internet mail extensions) multipart/related content type. MIME is used as
a packaging solution because of the diverse nature of information exchanged between
partners in e-business environments. The specification is under construction.

Note, recent news shows that ebXML has determined to integrate SOAP into its
- The deep color classes belong to Business Collaboration

**Figure 13** UML class diagram of Business Collaboration in ebXML (ebXML CMG, 2001)
The deep color classes belong to Business Transaction

Figure 14  UML class diagram of Business Transaction in ebXML (ebXML CMG, 2001)
**Figure 15** UML class diagram of Document Definition in ebXML (ebXML CMG, 2001)
- The deep color classes belong to Business Transaction

Figure 16  UML class diagram of Choreography in ebXML (ebXML CMG, 2001)
<!ELEMENT binary-collaboration ( ( documentation | business-transaction-activity | collaboration-activity | syn-state | start | transition | success | failure ) *)>

<!ATTLIST binary-collaboration
  name CDATA #REQUIRED
  initiator CDATA #REQUIRED
  responder CDATA #IMPLIED
  timeUnit (milliseconds | seconds | minutes | hours | days | weeks | months | years ) "minutes">

**Figure 17**  DTD of binary-collaboration corresponding to BinaryCollaboration in UML class diagram in Figure 13 (ebXML CMG, 2001)
Figure 18  The messaging service architecture in ebXML (ebXML TATP, 2001)
The ebXML message structure (ebXML TAPT, 2001)

**Figure 19** The ebXML message structure (ebXML TAPT, 2001)
3.2 eCo Framework

eCo Framework is the first to offer the specifications of XML-based registry mechanism in e-commerce. It is much earlier than ebXML and UDDI (see Figure 10).

eCo Framework provides a systematic way to view e-commerce systems from seven layers: Networks, Markets, Businesses, Services, Interactions, Documents, and Information Items (see Figure 20). Each layer of an eCo-compliant e-commerce system presents information about itself. By examining this information, others can locate the system; understand what it is for; recognize what market(s) it participates in; identify protocols the system uses to communicate; discover what documents the system uses to conduct business; learn how to interoperate with the system. Thus, the layers that can be accessed to answer such business and systems questions as:

- What other businesses can I find?
- What services do they offer?
- What kinds of interactions do they expect?
- What protocols do they follow?
- Can our systems communicate?
- What application interfaces do they provide?
- Are the interfaces compatible?
- What information must we exchange?

Table 4 shows the interaction of buying a computer in the seven-layer model.

The information items or properties of each layer are described and separated through various “Type Registries”. Each Registry can offer information on one or more specific sets of types appropriate to that Registry. For example, the Business Registry provides a set of type definitions required by the Business Layer, the Interaction Registry provides a set of Interaction types and Message Container types.

The Documents Layer and the Information Items Layer directly concern the issue of semantics for interoperability. The layers define both document types and the types of the contents of documents, the information items according to DTDs and the XML schema: XDR (XML Data Reduced) and SOX (Schema for Object-oriented XML)\(^7\).

\(^7\) XDR (XDR 1998) and SOX (SOX, 1999) are the earlier published and most popular versions of XML schema languages. XDR is mainly proposed by Microsoft, and SOX is proposed by Commerce One Inc. Currently the XML Schema we mean is managed by W3C (http://www.w3.org/XML/Schema). It has been advanced to Proposed Recommendation status since 2001-03-16.
Figure 20  The relationships of seven layers in eCo Framework (eCo, 1999)
Table 4  Buying a computer as an example in a seven-layer architecture (eCo, 1999)

<table>
<thead>
<tr>
<th>eCo Layer</th>
<th>Client System</th>
<th>Registry or Provider System</th>
</tr>
</thead>
<tbody>
<tr>
<td>Network Layer</td>
<td>Where can I get a computer?</td>
<td>Our markets are: computers, phones, copy machines…</td>
</tr>
<tr>
<td>Market Layer</td>
<td>Great, who sells computers?</td>
<td>Selling businesses are X, Y, Z…</td>
</tr>
<tr>
<td>Business Layer</td>
<td>I like Y, can I order?</td>
<td>From Y, you can buy or lease.</td>
</tr>
<tr>
<td>Service Layer</td>
<td>How can I order?</td>
<td>Y knows OFX, OBI, XML.</td>
</tr>
<tr>
<td>Interaction Layer</td>
<td>Here is my order.</td>
<td>Thanks! Here’s your receipt.</td>
</tr>
<tr>
<td>Document Layer</td>
<td>Meanwhile, behind the scenes, the transaction is completed automatically by the buying and selling systems that can now interoperate using the eCo Specification.</td>
<td>The style, structure and content of the receipt are presented to the server.</td>
</tr>
<tr>
<td>Data Element Layer</td>
<td></td>
<td>Data listing quantity, specification and shipping information are automatically accessed from the purchase order and inventory database and entered into the receipt and other documents essential to the transaction.</td>
</tr>
</tbody>
</table>
3.3 UDDI

3.3.1 Overview

Led by Ariba, IBM and Microsoft, UDDI is a cross-industry initiative designed to create a platform-independent, open framework for describing services, discovering businesses, and integrating business services using the Internet. Registry is the main building block that will enable businesses to quickly, easily and dynamically find and transact business with one another using their preferred applications.

At the high level, the standard defines three kinds of data services:

- **White Pages (general information)** contains business names, descriptions of the type of business, and other information regarding what kinds of services a vendor uses and also what technology they can respond to.

- **Yellow Pages (business categories)** adopts current government codes for tagging types of business operations as well as international and technology-based naming protocols. Besides, the Yellow Pages classify companies by geographical location.

- **Green Pages (how business is conducted)** provide more specific information on what types of documents a company can receive, the entry points for transactions and the technology they currently interact with and support.

3.3.2 Registry Operator

The UDDI Registry is run as a Web service or Operator that supports the UDDI specifications. Operators are companies that will run instances of the Business Registry. The operators will replicate the registrations across all nodes on a regular basis thus resulting in a complete set of registered records available to all. The operators support a common set of APIs that will ensure the integrity and availability of the information provided. Initially, Ariba, IBM and Microsoft operate this distributed service (Operator) and *for the next year and a half, at that point, an as-yet unnamed standards body will oversee the project* (Vance 2000).

3.3.3 Cost to register and assess to the Registry Operator

The cost to register a business depends on each Operator. Each Operator makes an independent decision whether to charge for registering or not. Ariba, IBM and Microsoft have each decided not to charge a registration fee. In future implementations of the UDDI specification, a certificate from a third party may be required for registration that may require a fee. The cost to access to a UDDI Registry is free and offered by Ariba, IBM and Microsoft.

3.3.4 UDDI APIs

Figure 21 demonstrates the information that a business can register. The information includes several kinds of simple data that help others determine the answers to the question, who, what, where and how:
• Who? businessEntity such as name, business identifiers, and contact information.
• What? businessService, the classification information that includes industry codes and product classifications, as well as descriptive information about the services that are available for electronic interchange.
• Where? bindingTemplate, the URL or email address through which each type of service is accessed.
• How? tModel, information about specifications that describe how a particular software package or technical interface functions. tModel provides software configuration information by way of tModel entries and tModel references that allow compatible services to be easily identified and traced.

The programming APIs in UDDI (Boubez et al 2000) are simple to understand. They are summarized in Figure 22. The interaction between the core UDDI elements is illustrated in Figure 23. This UML class diagram is corresponding to ones of ebXML interaction modeling (Figures 13-16). Any company hoping to join in a UDDI operation site can register (update), delete, login, logout and query the UDDI site. For example, to ‘register’, there are four APIs with the pattern of save_XXX:
• save_business, used to register new businessEntity information or update existing businessEntity information. This is used to control the overall information about the entire business.
• save_service, used to register or update complete information about a businessService presented by a specified businessEntity.
• save_binding, used to register new bindingTemplate information or update existing bindingTemplate information. This is used to control information about technical capabilities presented by a registered business.
• save_tModel, used to register or update complete information about a tModel.

After you register your services, you can delete them via the APIs with the pattern of delete_XXX. The interfaces of get_authToken and delete_tModel are used to request and cancel the authentication tokens required to use all other APIs. Then you can query your own business registry and others using two patterns of find_XXX and get_XXXDetail.

To help understand how to use the APIs, I cite the interface of find_business as an example to show its usage. Its syntax is as follows:

```xml
<find_business generic="1.0" maxRows="" xmlns="urn:uddi-org:api">
  [<findQualifiers/>]
  <name/> | <identifierBag/> | <categoryBag/> | tModelBag/> | <discoveryURLs/>
</find_business>
```

The arguments are as follows:
• generic, UDDI API version.
• maxRows, maximum integer of the number of results returned.
• xmlns, UDDI namespaces.
• findQualifiers, used to alter the default behavior of search functionality.
• name, a string representing a partial name.
• identifierBag, a list of business identifier references.
• categoryBag, a list of category references.
• tModelBag, used to search for businesses that have bindings that are compatible with a specific tModel pattern.
• discoveryURLs, a lot of URLs to be matched against the data associated with the discoveryURL’s contents of registered businessEntity information.

The expected return is a businessList with zero or not zero. Otherwise warning information will be given.

![Diagram of UDDI hierarchy and key XML elements]

Figure 21  The information hierarchy and the key XML elements in UDDI (Boubez et al, 2000)
XXX can be replaced by any UDDI data type: **businessEntity**, **businessService**, **bindingTemplate** or **tModel**.

**Figure 22** The APIs used to register in a UDDI operator
The relationships between the core elements in UDDI

Figure 23  The relationships between the core elements in UDDI
3.4 A comparison of The frameworks

3.4.1 Main objective
The objective of ebXML is to exploit both EDI’s past investment and XML’s technology capabilities. It not only adopts a collection of EDI business processes and translates EDI’s message format into XML, but also provides a huge infrastructure to support the Internet. It attempts to include all XML applications for conducting e-business.

The eCo Framework has nearly the same goal as ebXML. It focuses on a systematic view of electronic commerce systems in seven layers and the semantic development of XML documents in multiple situations.

UDDI concentrates on creating some distributed registry sites and providing some APIs thereby allowing any sized companies to quickly and inexpensively register and access to the sites.

3.4.2 Architecture
The ebXML and the eCo Framework provide architectural design from top to bottom. First they regard e-commerce as one whole system and they believe they can offer such a framework to let all kinds of organizations and communities sit together under their roof as so to achieve the interoperability between trading partners. After setting the background of e-commerce, they gradually implement the specifications of the modules constituting the system one by one.

UDDI takes architecture from the most important part, neither from top to bottom nor from bottom to top. To establish a large e-commerce system, first of all it needs to perform registry and discovery.

3.4.3 Registry in ebXML and UDDI
From the objective and architecture discussed above, we easily realize that ebXML and UDDI are comparable with respect to registry. We compare them from four perspectives:

- **Overview.** UDDI is to offer White, Yellow, and Green Pages in some sites to hold business and service information. ebXML is to provide a Registry Service as well as a Repository Service to store business and service information and related specifications such as vocabulary specifications.

- **How to start message exchange between trading partners.** To use UDDI registry, a trading partner has to register its bindingTemplate so as to use the URL or email addresses to access to Registry. ebXML users have to register a CPP (Collaboration Protocol Profile) so that business partners find its services via the service interface in CPP.

- **How to get software configuration information of Web services.** UDDI uses a tModel to define software configuration data whereas ebXML uses a CPA (Collaboration Protocol Agreement) to get configuration information.
• **Message request/response and message structure.** UDDI uses SOAP-based APIs for registration and search. ebXML also selects SOAP as messaging mechanisms before defining general messaging services to fit any situations (ebXML 2001).

• **How to share information between different registries.** UDDI offers a protocol for replication between Registry operators and agreements to operate and replicate are signed in advance. ebXML uses a distributed approach or publish/subscribe mechanism without replication.

• **Categorization.** Both need a categorization system to store business and service information in their Registry. UDDI defines its own methods specific for its applications while ebXML is to define a rather general and standardized one.

### 3.4.4 Marketing strategy

The success of the frameworks finally depends on the extent to which they can be accepted and adopted by companies. The strategy of these frameworks to capture customers is different. Because eCo Framework has quit the competition in 1999, here I just discuss UDDI and ebXML.

The strategy of UDDI is quick participation and quick implementation. UDDI, together with WSDL and SOAP, is designing a route for those businesses that wish to join in the venture of e-commerce to easy access. This fast route includes the following ideas:

• IBM, Microsoft and Ariba are leading the UDDI project, and their great influence has obviously made it widely advertised. In fact many big companies have had to become followers of UDDI although they are reluctant, e.g., Oracle (Scannell and Sullivan 2000), Hewlett Packard and Intel (Vance 2000).

• Fast access is possible. The UDDI offers WSDL for describing Web services, SOAP for transporting XML documents, and Programming APIs to implement the interfaces to interact with UDDI operation sites.

• Inexpensive entry cost is necessary. What you should invest in are some toolkits for UDDI, WSDL and SOAP (most of them are open source), and advanced enterprise servers (you normally have them even though you do not register your Web services).

• An inexpensive departure cost is incurred. You can quit the project whenever you want.

• Visible benefit is obvious. The first stop of this route is UDDI. If you implement a UDDI business registry, you at least make your products and services available to the whole world, thus you may gain some potential market opportunities. This obvious gain is welcome even though you cannot arrive at the next stop.

Compared to that of the UDDI, the strategy of the ebXML is vendor-neutral and complete architecture.

• UN/CEFACT and OASIS lead the project, and the specifications tend to be neutral and less affected by vendors because both are non-profit organizations. Many organizations can warmly welcome this feature. For example, some toolkit software developers like this approach when the forming business opportunities are occupied by the leading companies.
A complete architecture lays a solid background for international e-commerce. This basic framework is beneficial for global economic development.

Obvious benefit may occur for current EDI users but how to collect the benefit is under construction. Business processes in EDI can easily be integrated with Web services.

3.4.5 Disadvantages

We still just consider ebXML and UDDI in this section. The biggest advantage of ebXML is its complete architecture for international e-commerce; at the same time it also has created a strong burden to perform quick implementation because it is subject to more constraints and interests of various organizations. For example we have not seen any obvious efforts on semantics and financial processes (although payment is one important business process).

The advantage of the UDDI is its quick implementation, quick access and broad industrial support. But you have to consider the following issues:

- Business independence. Once you enroll in one UDDI Operator, what is the extent to which you depend on it and the software the Operator offers?
- UDDI’s openness. Is the base technology of UDDI open or proprietary to some main players?
- Cost to register and access to a UDDI Operator.
- Future development, e.g., the leaders of UDDI claim to send UDDI registry to a standard body, what standards body the UDDI Registry will be delivered?

3.4.6 Maturity

Figure 10 shows the different stages of the three frameworks. eCo Framework has wrapped up in 1999.

The specifications of ebXML are under development so far. The only proven specifications have Technical Architecture Specification Version 1.0.4 (010216) and Requirements Specification Version 1.0 (000512). The complete specifications will be released in March 2001 (Meehan 2000) or later in May 2001. There is still a long way to go if you want to use it in your project because such requirements as vocabulary, syntax and semantics cannot be met in such a short time.

UDDI addresses much smaller issues in e-commerce. As for their similar function as to business registry, UDDI is more mature than ebXML with regard to the specification, Programming APIs, supporting tools, and the degree of acceptance.

Both are quickly updating and the race has not ended yet.
3.5 Future development

The collaboration between ebXML and UDDI is unavoidable.

- From the perspective of functionality, both aim to fulfill the interoperation of e-business, UDDI targets at business registry, and the ebXML at registry, trading partner agreement, and transport/routing/packaging.
- From the latest development of ebXML, ebXML is taking measures to combine with UDDI and thus increase the industrial support to its specification. For instance, ebXML is determined to integrate SOAP with messaging services (ebXML 2001), and ebXML provides specific interfaces to connect to UDDI Operators.
- They are complementary in terms of winning wide acceptance. UDDI offers a quick implementation to join in an e-business venture, but later on it needs to find ways of adding more functions such as collaboration and contract. ebXML will greatly compensate for the requirements UDDI demands. In the meantime, ebXML requires more implementation testing in a broad range that UDDI can just offer.
- Both have same requirements to some degree. For example, both demand a common accepted vocabulary, syntax, semantics and industrial support. Combining these will be beneficial to lay a solid foundation to XFIC.

But how do they consolidate? In my opinion, ebXML as a non-profit organization will continue to focus on standardization of XML on aspects of vocabulary, syntax, semantics, messaging and business collaboration. Three factors are most important:

- Time should be as brief as possible.
- The complexity of implementation should as simple as possible.
- Industrial standards or industrially accepted methods should be adopted in ebXML.
- Quality is the key for ebXML to survive in the future.

UDDI should continually focus on quick implementation combined with the specifications ebXML provides. Two factors are most important:

- Openness is necessary. Openness at least includes open source software to support Registry and Search, open entry and accession, open voting for major decisions.
- Commercial benefits should be obvious. It is often measured through ROI (Return on investment).
This chapter focuses on those frameworks which emphasize XML messaging. Here we assume that two applications are specified. Under this circumstance, we will choose such frameworks as SOAP, BizTalk, cXML, ICE, OAG, RosettaNet, OFX, VoiceXML, WSDL, RDF and xCBL. The intention of this chapter is to provide information on these frameworks one by one. This will help choose the right frameworks to quickly achieve interoperability.

4 XFIC: focus on XML messaging

This chapter focuses on those frameworks which emphasize XML messaging. Here we assume that two applications are specified. Under this circumstance, we will choose such frameworks as SOAP, BizTalk, cXML, ICE, OAG, RosettaNet, OFX, VoiceXML, WSDL, RDF and xCBL. The intention of this chapter is to provide information on these frameworks one by one. This will help choose the right frameworks to quickly achieve interoperability.

4.1 SOAP and BizTalk

4.1.1 SOAP (Simple Object Access Protocol)

SOAP is a lightweight protocol for realizing RPC (Remote Procedure call) via XML in a decentralized and distributed environment. “Lightweight” means that SOAP is an XML marshalling and unmarshalling mechanism and does not need heavy libraries like CORBA and DCOM.

The SOAP protocol consists of three parts (SOAP 2000):

- A SOAP envelope that defines a framework for describing what is in a message, who should deal with it, and whether it is optional or mandatory. Figure 24 illustrates a typical SOAP envelope. Figure 26 shows one SOAP message for the request/response model. You can compare it to Figure 2, where the element of Header does not exist, because it is optional and represents a generic mechanism for adding features to a SOAP message. Here we can see that SOAP is self-descriptive via its envelope.

- A set of encoding rules for defining a serialization mechanism to exchange instances of application-defined data types. SOAP encoding rules are encoded as an XML namespace declaration: http://schemas.xmlsoap.org/soap/encoding/.

- A convention for representing RPCs and responses. It includes binding HTTP to carry the URI of the target object, and modeling a method invocation as a struct, a compound type defined by SOAP.

Perhaps you want to know how XML messaging was performed before these frameworks were proposed. There are many solutions (Morgenthal and Forge 2000). For example, you can use the technology of serializing the state of Java Object graphs into XML and thus to allow Java and non-Java applications to communicate. You can also use JMS (Java Message Services) with XML to do it. However, all implementations are proprietary, and meet your integration goal and ignore integration with other systems.
HTTP has been bound with SOAP to transport XML documents (for two related examples see 4.1.2 and 4.6.1). The advantage of the binding is its ability to use the formalism and decentralized flexibility of SOAP with the rich feature set of HTTP. The combination of SOAP and HTTP allows SOAP to cross nearly all firewalls, because HTTP or HTTPS (HTTP Secure) is permitted to pass any firewalls.

However, HTTP is not the only solution to the SOAP transport method. SMTP can be used instead, Apache-SOAP implementation (http://xml.apache.org/) achieved SMTP mechanism as well as HTTP. RMI can also be an optional transport mechanism. Govindaraju et al (2000) did an experiment in scientific computing using a binding of RMI and SOAP, and showed the fast speed for small messages and great value for multi-protocol environments.

The main objective of SOAP is to achieve interoperability by using XML to interact with interoperable Web services. SOAP addresses interoperability at the data serialization level, allowing disparate distributed programming systems to interact without disrupting the features of each platform. This characteristic makes SOAP a Web services language. UDDI and WSDL have adopted it and the BizTalk framework is an application of it. Many large companies support this protocol, such as Microsoft and IBM.

![Transport Envelop](image)

**Figure 24** The structure of SOAP message
4.1.2 BizTalk

BizTalk was initiated by Microsoft to enable applications to securely and reliably exchange XML documents. BizTalk starts from SOAP to build its document structure and transport mechanism. The purpose of BizTalk Framework is how to quickly implement SOAP messaging. What you need is to acquire a BFC (BizTalk Framework Compliant) server, capture SOAP knowledge, and abide by the BizTalk conventions involved.

BizTalk provides detailed specifications for the construction of BizTalk Documents and Messages and their secure transport over a number of Internet-standard transport and transfer protocols (Microsoft 2000). BizTalk Documents follow a number of rules for structure and content in order to provide rich functionality and predictable semantics. A BizTalk specification describes the following aspects of BizTalk Documents and their semantics:

- Overall structure of BizTalk Documents.
- BizTalk headers for document routing, properties, catalog, and process management.
- Structure and handling of BizTalk Documents that require reliable delivery.

When implementing solutions using the BizTalk Framework, specific transport methods, encoding, and security mechanisms must be used to secure and deliver messages. BizTalk describes the following mechanisms and aspects of BizTalk Message encoding and transport:

- Transport bindings for Internet transfer protocols
- MIME-based transfer encoding and attachment packaging
- Signatures and encryption based on secure MIME

The logical implementation model for the BizTalk Framework is composed of three layers: the application, BFC server and transport (Figure 25)

- The application (and appropriate adapters) is the ultimate source and destination of the content of BizTalk Messages, and communicates with other applications by sending Business Documents back and forth through BFC Servers.
- A BFC server is represented by the set of services providing the messaging-processing functionality. Such services include document routing and transformation with high scalability, robustness and security.
- Multiple BFC Servers communicate with one another over a variety of protocols, such as HTTP, SMTP, and Microsoft Message Queue (MSMQ). The BizTalk Framework does not prescribe what these transport protocols are, and is independent of the implementation details of each.

The application is responsible for generating the Business Documents and any attachments to be transmitted to its peer(s) and submitting them to the BFC Server. The responsibility for wrapping the Business Documents in a BizTalk Document may
rest with either the application or the BFC server, depending on the implementation of the BFC server.

The server processes the document and any attachments and constructs a BizTalk Message as appropriate for the transport protocol. The BFC Server uses information contained in the BizTags to determine the correct transport-specific destination address. The server then hands the message to the transport layer for transmission to the destination BFC Server. The interfaces between the business application, the BFC Server, and the transport layer are implementation-specific.

This BizTalk Document consists of a standard SOAP 1.1 message that contains the two parts (Figure 26):

- BizTalk-specific <endpoints> and <properties> SOAP header entries, constructed using BizTags defined in standard BizTag namespaces (http://schemas.biztalk.org/btf-2-0/).
- An application-specific Business Document (in this case a book purchase order), with its own application-defined XML namespace (http://electrocommerce.org/purchase_order/), carried in the body of the SOAP message.

Some points are further explained about Figure 26:

- SOAP-ENV: Envelope has three attributes to define three namespaces:
  - xmlns:SOAP-ENV\(^9\) represents the versioning model and is mandatory in SOAP document. If an application receives a message with a different namespace, the application must treat this as version error and discard the message.
  - xmlns:SOAP-ENC, optional, indicates the serialization rules\(^10\) used in SOAP.
  - xmlns:xsi, optional, is an instance namespace as defined by XML Schema Definition (XSD) (http://www.w3.org/1999/XMLSchema/). This example uses xsi:type two times to declare address type. As address’s attribute, xsi:type is used to define the type of address so that the receiving application can deserialize address in the method call (or response).

- SOAP-ENV: Envelope consists of two elements: Header (optional) and Body (mandatory).

- SOAP-ENV: mustUnderstand is used to indicate whether a header entry is mandatory or optional for the recipient to process. Its value is either “1” or “0” (default). A value of “1” means that the recipient of that header entry either must

---

\(^9\) xmlns:SOAP-ENV can be written as SOAP-ENV to let SOAP-ENV be the default namespace of SOAP document, thus you do not need to add SOAP-ENV before related elements such as Header and Body in Figure 25.

\(^10\) Serialization rules address how to encode the values to let them pass the network like XDR (eXternal Data Representation, a standard to encode the values in the RPC call and reply messages). The SOAP encoding style is based on a simple type system that is a generalization of the common features found in type systems in programming language, databases and semi-structured data.
obey the semantics (the fully qualified name of the element) and process according to those semantics, or must fail processing the message.

- **endpoints** and **properties** are two mandatory elements in BizTalk-specific SOAP documents.
  - **Endpoints** indicates the document source and destination. It includes two mandatory elements, **to** (destination business address) and **from** (source business address). Note that the addresses are business-related namespaces, not network transport. It is entirely possible that multiple transports are available for communication and that they will change over time. A SOAP document is independent of network transport mechanisms.
  - **Properties** indicates document identification and properties. It includes four mandatory elements: **identify** (URI reference for the purpose of logging, tracking, error handing or other document processing correlation requirements), **sentAt** (sending timestamp), **expiresAt** (the expiration timestamp), and **topic** (a URL reference to uniquely identify the overall purpose of the BizTalk document).

![Figure 25](three-layer-biztalk-implementation-model.png)  
*Figure 25*  Three-layer BizTalk Implementation Model (Microsoft 2000)
Figure 26  A Typical BizTalk Document: Purchase Order
4.2 Specific business operations

4.2.1 cXML (commerce XML)

A joint effort of more than 40 companies spearheaded by Ariba Inc., cXML is an open, Internet-based standard designed to facilitate easy exchange of catalog content and transaction information between trading partners (Shim et al 2000). Simplicity and quick implementation for e-procurement systems are two characteristics.

There are two basic communications modes for cXML transactions: request/response and one way (Ariba 2000). Request/response transactions can be performed only over an HTTP connection. Figure 27 illustrates the interaction between entity A and entity B. One-way transaction is asynchronous, which means:

- A does not wait for the response from B.
- B does not need to send back the reply after B processes the cXML document.

The format of cXML documents is similar to SOAP (see Figure 26). It consists of two parts:

- Header contains authentication information and addressing.
- Request or Response data contains a specific request or response and the information to be passed.

Figure 28 shows one typical request order including transport information of this cXML document in **Header**, and payload in **Request**. Request has just one element **OrderRequest** that includes **OrderRequestHeader** and **ItemOut**. **OrderRequestHeader** consists of the information about the total orders: **Total** (total payment), **ShipTo** (the shipping address), **BillTo** (the billing or invoicing address), and **Payment** (how to pay). **ItemOut** includes detailed order information: its attribute **quantity** (how may), **ItemDetail** (price, description, etc), and **Distribution** (how to pay for distribution).

Figure 29 is the response document that acknowledges that the recipient received the purchase order and that it parses correctly, i.e., it is a valid cXML document. However, this is not a commitment to execute the purchase order. Here, **Response** uses **Status** and its attribute **code** and **text** to express response information, which may include 200/OK, 201/Accepted, 204/No Content, 400/unauthorized, 402/Payment Required, 403/forbidden, 406/Not Acceptable, 409/conflict, 412/Precondition Failed, 417/expectation failed, 500/internal server error, and 501/not implemented. These status codes are according to HTTP 1.1 specification because cXML is bound to HTTP as transport mechanism. In addition, cXML includes three non-HTTP status codes:

- 550: unable to reach nest cXML server to complete a transaction requiring upstream connections.
- 551: unable to forward request due to supplier misconfiguration.
- 560: temporary server error.
The transaction processes are described as follows:
1. A establishes an HTTP/1.x connection with B via URL of B.
2. A uses a POST operation to send the cXML document through HTTP connection.
3. A waits for a response to return through HTTP connection.
4. B has an HTTP/1.x-compliant server that dispatches the HTTP request to the URI resource via script language.
5. B’s resource identified in step 4 reads the cXML document contents and maps the Request to the appropriate handler for that request.
6. B’s handler for the cXML Request performs the work that the Request specifies and formats a cXML document as a Request.
7. B sends the cXML Response to A through the HTTP connection established in step 1.
8. A reads the cXML Request and returns it to the process that initiated the Request.
9. A closes the HTTP connection established in step 1.

Figure 27  Request/Response Model between entity A and B in cXML
A typical customer order in cXML

Figure 28
The Response to the customer order request in Figure 28

**Figure 29** The Response to the customer order request in Figure 28

### 4.2.2 ICE (Information and Content Exchange)

ICE is an initiative hosted by IDEAlliance, formed by the Graphic Communications Association (GCA). It is used to define the roles, responsibility and XML document formats of syndicator (a content aggregator and distributor) and subscriber. It is a request/response model, but it focuses on 1-to-n information exchanges. Two simple scenarios are used, for example, in syndication of news headlines from an online publisher to other online services, and syndication of a parts catalog from a manufacturer to its distributors. Such purpose can fulfill many requirements of business-to-business marketplaces.

The Subscriber and Syndicator assume three roles during the ICE operations:
1. Subscriber/Syndicator with respect to the business relationships where the Syndicator distributes content to the Subscribers
2. Requester/Responder based on who initiates the ICE operation. The initiator is the Requester, and the other party, who performs the operation, is the Responder.
3. Sender/Receiver based on the transmission of a single payload, an XML message.

The ICE protocol covers four general types of operations:
1. Subscription Establishment and Management
2. Data Delivery. A syndicator pushes content to a subscriber or a subscriber pulls content from a syndicator.
3. Event logs. ICE allows a Syndicator to request the protocol event logs of the Subscriber, and vice versa, as an aid for debugging and diagnosis.
4. Miscellaneous. Miscellaneous operations include the ability to renegotiate protocol parameters in an established relationship, the ability to send unsolicited ad-hoc notifications (i.e., textual messages) between systems (presumably ultimately targeted at administrators), the ability to query and ascertain the state of the relationship, etc.

A typical sequence of Subscription Establishment and Management events is:
1. A user (technical manager, engineer, etc.) at the Syndicator site creates a new Subscriber account using the ICE software on the Syndicator's system. This
operation is not defined by the protocol; it is a property of the tools used by the Syndicator.

2. The Syndicator tells the Subscriber what URL to use for ICE communication. It is likely that this URL will be under access control, and the Syndicator (human) will communicate the necessary authentication data to the Subscriber (human) using an out-of-band mechanism.

3. A user (technical manager, engineer, etc.) at the Subscriber site will enter the necessary data into the Subscribers ICE system.

4. ICE protocol operations are now ready to begin: the Subscriber will authenticate (if necessary) to the given URL and issue the first ICE request: a request for a catalog.

5. The Syndicator will return a catalog containing subscription offers to which the Subscriber is entitled to subscribe. These were most likely set up as part of the creation of the Subscriber account performed in step 1.

6. The ICE tools on both ends negotiate (in the sense of choosing protocol parameters) a mutually agreeable set of parameters for a subscription. For example, both sides may have a preferred set of delivery times.

7. A subscription is established and packages can begin to be exchanged.

Negotiation in step 6 is a special function that other frameworks do not directly address. Typically, a Subscriber will use ice-get-catalog to get a catalog, take one of the ice-offers from that catalog, prioritize the parameters and send it back to the Syndicator in a request. However, the Subscriber is free to create an ice-offer in any implementation-defined manner it wants. After the Subscriber sends the offer to the Syndicator, the Syndicator can respond in one of Sorry, OK or Error.

An example is modified from Souzis et al (2000) shown in Figure 30-33. Joe Cool, a Subscriber, first requests a catalog from www.BradsGadgets.com using an XML document (see Figure 30). A catalog response is shown in Figure 31. Detailed catalog information is shown in ice-catalog, whose attributes name and url indicate the information source. Inside ice-catalog is ice-contact to describe contact information and multiple ice-offer in the catalog.

After a catalog response is received, a subscription XML document is encoded (see Figure 32). Sometimes Joe would like to cancel the subscription and Syndicator has to confirm the subscription using the document shown in Figure 33. We can see that each role has its own sender-id (Joe is 4af37b30-2c35-11d2-be4a-204c4bf5020, the syndicator is 4a2180c9-9435-d00f-9317-204d974e34f0) and each offer has its own offer-id, and these identifiers link the payload together. Figure 32 shows mode="pull" in element ice-delivery-rule, meaning that Joe pulls the content of Digital Cameras 2000 Newsletter from the syndicator.
<?xml version="1.0"?>
<!DOCTYPE ice-payload SYSTEM "http://www.icestandard.org/dtds/ICE1_1.dtd">
<ice-payload payload-id="PL-2000-08-24T22:10:33.901-JoeCool423" timestamp="22:10:33.741" ice.version="1.1">
  <ice-header>
    <ice-sender sender-id="4af37b30-2c35-11d2-be4a-204c4f50201">
      name="Joe Cool" role="subscriber" />
    <ice-user-agent>IceBlock Systems ICE Processor, V7.0</ice-user-agent>
  </ice-header>
  <ice-request request-id="2000-08-24T22:10:33_RQ_JoeCool_1673">
    <ice-get-catalog />
  </ice-request>
</ice-payload>

Figure 30  XML document to request a catalog in ICE
Figure 31  The response of ice-catalog to the request shown in Figure 30
Figure 32 The Subscriber subscribes to the Syndicator’s offer shown in Figure 31
Figure 33  The Subscriber cancels its subscription and Syndicator confirms it following Figure 32
4.3 Enterprise application integration (EAI)

Integration is one of the biggest problems related to Internet commerce, because we have to face daunting challenges—disparate legacy systems; a hodgepodge of hardware, operating systems, and networking technology; proprietary packaged applications; and more. EAI offers a solution to this increasingly urgent business need. It encompasses technologies that enable business processes and data to speak to one another across applications, integrating many individual systems into a seamless whole (Linthicum 2000). Many industrial projects are initiated to use XML to attack the problem. OAG (Open Applications Group), RosettaNet and Wf-XML (Workflow XML) are three of them.

4.3.1 OAG (Open Applications Group)

OAG is a non-profit consortium focusing on best practices and process-based XML content for e-business and application integration. It is the largest publisher of XML-based content for interoperable enterprise component software in the world. It also emphasizes the simplicity with aims to provide the common DTDs for several types of industry. ERP vendors such as SAP, J. D. Edwards, and PeopleSoft are its members.

OAG also stresses the model of request/response implemented via connectors or adapters defined by interface specifications. To meet the requirements, OAG proposes three concepts:

- Business object document (BOD), containing control area and business data area, similar to the message structure of BizTalk, cXML or ICE. Figure 34 shows a COMFIRM_BOD XML document. The control area (CNTROLAREA) includes the purpose of sending this XML document, the sender information and sending time. Pay attention to VERB and NOUN that define the service request action and the object to be performed. The body area (DOCUMENT) represents the payload to confirm the order from Yuxiao Zhao.

- OAG Interface Specification (OAGIS) to describe the process for development, translation, and implementation of BOD through resource DTDs and an additional DTD for each service request. The resource DTDs include information that is common across all requests. Resource DTDs are used to define data types, fields and segments. A small number of complex data types, or super segments are also defined (OAG 2000).

- OAG offers over 50 detailed integration scenarios such as Order Management to Accounts, Purchasing to Accounts Payable, and Manufacturing to Purchasing. Each scenario consists of workflow, assumptions and exceptions requirements. In fact OAG attempts to recommend all integration scenarios involving all enterprise systems! Figure 35 shows the integration scenario from ERP to finite scheduling and MES (Manufacturing Execution System). Again, note that all requests and responses are expressed with a VERB and a NOUN (OAG 2000).
<?xml version="1.0"?>
<!DOCTYPE confirm_bod_002 SYSTEM "002_confirm_bod_002.dtd">
<BOD>
  <CNTROLAREA>
    <BSR>
      <VERB>CONFIRM</VERB>
      <NOUN>BOD</NOUN>
      <REVISION>002</REVISION>
    </BSR>
    <SENDER>
      <LOGICALID>XXX1234YYY</LOGICALID>
      <COMPONENT>G/L</COMPONENT>
      <TASK>CONFIRM</TASK>
      <REFERENCEID>REF1</REFERENCEID>
      <CONFIRMATION>0</CONFIRMATION>
      <LANGUAGE>ENG</LANGUAGE>
      <AUTHID>JOE DOE</AUTHID>
    </SENDER>
    <DATETIME qualifier = "CREATION">
      <YEAR>1995</YEAR>
      <MONTH>12</MONTH>
      <DAY>31</DAY>
      <HOUR>17</HOUR>
      <MINUTE>59</MINUTE>
      <SECOND>0</SECOND>
      <SUBSECOND>000</SUBSECOND>
      <TIMEZONE>-0500</TIMEZONE>
    </DATETIME>
  </CNTROLAREA>
  <DOCUMENT>
    <CONFIRM_BOD>
      <CONFIRM>
        <SENDER>
          <LOGICALID>XX141HG09</LOGICALID>
          <COMPONENT>INVENTORY</COMPONENT>
          <TASK>RECEIPT</TASK>
          <REFERENCEID>95129945823449</REFERENCEID>
          <CONFIRMATION/>
          <LANGUAGE>ENG</LANGUAGE>
          <AUTHID>Yuxiao Zhao</AUTHID>
        </SENDER>
        <STATUSLVL>00</STATUSLVL>
        <DESCRIPTN>PROCESSED WITHOUT ERRORS</DESCRIPTN>
        <ORIGREF>RCP7#12550699</ORIGREF>
      </CONFIRM>
    </CONFIRM_BOD>
  </DOCUMENT>
</BOD>

Figure 34  A typical XML document in Business Object Document in OAG
Figure 35  The Scenario to integrate ERP to finite scheduling and MES (OAG 2000)
4.3.2 RosettaNet

RosettaNet aims at integration between supply chain partners, transaction consistency and the elimination of inefficiencies such as phone, fax and email communications. Although RosettaNet focuses on Electronic Component, Information Technology and Semiconductor Manufacturing companies, its specifications can be applied in any industry (Shim et al, 2000). It streamlines business transactions by providing guidelines for trading partners in the supply chain. These guidelines are called Partner Interface Processes (PIPs) that are designed to define all business processes that occur between supply chain companies, and to provide the models and documents for the implementation of standards. Each PIP includes:

- Class and sequence diagrams analogous to workflow in OAG. Figure 36 is the activity diagram used for a customer (business partner) to Query Price and Availability (PIP3A2), Quote and Order Entry (PIP3A), Order Management (PIP3). Two interfaces, "<QueryResponseActivity>" and "<SecureFlow>" are used.
- XML document(s) based on DTDs, specifying PIP Services, Transactions, and Messages, which include dictionary Properties. Figure 37 shows the simplest query/response pattern used for Query Price and Availability shown in Figure 36. Note that "Available Quantity, Product Quantity", the "Monetary Amount" and the "Global Product Unit Of Measure" are intentionally left empty, instructions for the responding party to “fill-in-the-blanks”.
- Implementation guidelines that provide common exchange protocols to implement integrated components via network.

RosettaNet Dictionary provides a common set of properties for PIPs. It consists of business and technical dictionaries. Business dictionary designates the properties used in basic business activities, while the technical dictionary provide properties for defining products.

Currently, RosettaNet PIPs encompass six types (except RosettaNet Support):
1. Partner, Product and Service Review;
2. Product introduction;
3. Order Management;
4. Inventory Management;
5. Marketing Information Management;
6. Service and Support;
7. Manufacturing.

Like other frameworks, a PIP message comprises three parts: the Preamble Header, the Service Header, and Service Body, all of which need to be validated against RosettaNet-provided DTDs. The transportation mechanisms can be HTTP/POST, MIME and CGI (Common Gateway Interface).

---

11 XML and DTD are not available unless you are a RosettaNet member.
Figure 36  Query Price and Availability (PIP3A2) in RosettaNet

Price Availability Query
   Price And Availability Query
   Price And Availability
   Product Line Item
      Product Description
      Global Product Identifier “001234567890”
      Requested Quantity. Product Quantity “1000”
      Available Quantity. Product Quantity
      Unit Price. Financial Amount
      Monetary Amount
      Global Currency Code “USD”
      Global Product Unit Of Measure

Price Availability Response
   Product Price And Availability
   Product Line Item
      Product Description
      Global Product Identifier “001234567890”
      Requested Quantity. Product Quantity “1000”
      Available Quantity. Product Quantity “876”
      Unit Price. Financial Amount
      Monetary Amount “600”
      Global Currency Code “USD”
      Global Product Unit Of Measure “Each”

Figure 37  Query/Response design pattern in Query Price and Availability (PIP3A2)
4.3.3 Wf-XML (Workflow XML)

Wf-XML originally came from the SWAP (simple workflow access protocol) implementation in 1999. It represents the next step in the evolution of workflow interoperability standards (Hayes et al., 2000). Wf-XML is managed by the Workflow Management Coalition (WFMC).

At the high level, Wf-XML targets:

- Supporting chained and nested workflow.
- Providing for both synchronous and asynchronous interactions.
- Remaining implementation independent.
- Defining a light, easy-to-implement protocol.

To identify the primary groups of workflow for interoperability and request-response information between the groups, Wf-XML defines the Logic Resource Model (Figure 38). Three groups are identified as follows:

- The ProcessDefinition group is the most fundamental group of operations required for the interaction of generic services. It represents the description of a service’s most basic functions, and is the resource from which instances of a service will be created. Since every service to be enacted must be uniquely identifiable by an interoperating service of service requestor, the process definition will provide a resource identifier. When a service is to be enacted, this resource identifier will be used to reference the desired process to be executed.

- The ProcessInstance group represents the actual enactment of a given process definition and will have its own resource identifier separate from the definition’s. When a service is to be enacted, a requestor will reference a process definition’s resource identifier and create an instance of that definition. Since a new instance will be created for each enactment, the process definition may be invoked any number of times simultaneously. However, each process instance will be unique and exist only one time. Once created, a process instance may be started and will eventually be completed or terminated.

- The observer group provides a means by which a process instance may communicate upon completion or termination. In nested subprocesses, there must be a way for a requestor of a service enactment to determine or be informed when a subprocess completes. The Observer group will process this information by giving a process instance the resource identifier of the requestor.

The SWAP proposal attempts to define an Internet-based workflow access protocol to instantiate, control, and monitor workflow process instances. SWAP was envisioned as a binding of the jointFlow object model and related WfMC (Workflow Management Coalition) standards to an HTTP-based interaction protocol. The basic idea of the SWAP proposal-rendering interaction between components of workflow applications such as XML-encoded messages-provides an excellent base for adapting the WfMC standards in the area of message-based enterprise application integration. (Hayes, et al., 2000). More information can be found at www.ics.uci.edu/~ietfswap/.

---

12 The SWAP proposal attempts to define an Internet-based workflow access protocol to instantiate, control, and monitor workflow process instances. SWAP was envisioned as a binding of the jointFlow object model and related WfMC (Workflow Management Coalition) standards to an HTTP-based interaction protocol. The basic idea of the SWAP proposal-rendering interaction between components of workflow applications such as XML-encoded messages-provides an excellent base for adapting the WfMC standards in the area of message-based enterprise application integration. (Hayes, et al., 2000). More information can be found at www.ics.uci.edu/~ietfswap/.
Then the next task of Wf-XML is to define the request-response message structure for the operations (e.g., CreateProcessInstance, ProcessInstanceStateChanged, GetProcessInstanceData, and ChangeProcessInstanceState) and transport bindings. Consider a company Acme Manufacturing ordering a computer from Dell Computer Corporation. The request message of CreateProcessInstance is encoded in XML (Figure 39). Besides the element of WfTransport, it consists of two parts: WfMessageHeader and WfMessageBody. The response message (Figure 40) shows two alternatives. If the CreateProcessInstance operation is successful, then a resource identifier for the new process instance will be returned. Here a ProcessInstanceKey is given. Otherwise, an exception message is returned. Here error 502 represents WF_INVALID_PROCESS_DEFINITION, a kind of resource access error.

Transport bindings select HTTP-method POST and may use MIME body parts for input and output.

Three groups are identified as follows:

- **ProcessDefinition** creates an instance of a service.
- **ProcessInstance** is the actual enactment of a Process Definition.
- **Observer** allows a process to communicate status changes.

**Figure 38** Logic Resource Model in Wf-XML
<xml version="1.0"/>
<WfMessage>
  Version="1.0"
  xmlns="http://www wfmc.org/standards/docs/Wf-XML">
  <WfTransport/>
  <WfMessageHeader>
    <Request ResponseRequired = Yes >
      <Key>http://www.dell.com/Wfengine?id=1199827 </Key>
    </Request>
  </WfMessageHeader>
  <WfMessageBody>
    <CreateProcessInstance.Request StartImmediately = true >
      <ObserverKey>http://www.Acme.com/wfx456</ObserverKey>
      <ContextData>
        <Computer>
          <Type>notebook</Type>
          <Series>Inspiron</Series>
          <Number>7500</Number>
          <Option>DVD</Option>
        </Computer>
      </ContextData>
      </CreateProcessInstance.Request>
    </WfMessageBody>
  </WfMessage>
</WfMessage>

Figure 39  The request message (customer order) in Wf-XML
(1) ProcessInstanceKey is replied when the order shown in Figure 39 succeeds.

```xml
<?xml version= 1.0 ?>
<WfMessage Version= 1.0  xmlns="http://www.wfmc.org/standards/docs/Wf-XML">
  <WfTransport/>
  <WfMessageHeader>
    <Response/>
    <Key>http://www.dell.com/Wfengine?id=1199827 </Key>
  </WfMessageHeader>
  <WfMessageBody>
    <CreateProcessInstance.Response>
      <ProcessInstanceKey>
        http://www.dell.com/WfcXML1199
      </ProcessInstanceKey>
    </CreateProcessInstance.Response>
  </WfMessageBody>
</WfMessage>
```

(2) Error handling when the order shown in Figure 39 fails.

```xml
<?xml version= 1.0 ?>
<WfMessage Version= 1.0  xmlns="http://www.wfmc.org/standards/docs/Wf-XML">
  <WfTransport/>
  <WfMessageHeader>
    <Response/>
    <Key>http://www.dell.com/WfEngine?id=1199827 </Key>
  </WfMessageHeader>
  <WfMessageBody>
    <CreateProcessInstance.Response>
      <Exception>
        <MainCode>502</MainCode>
        <Type>F</Type>
        <Subject>Invalid Process Definition</Subject>
        <Description>Cannot create instance</Description>
      </Exception>
    </CreateProcessInstance.Response>
  </WfMessageBody>
</WfMessage>
```

**Figure 40** The response message in Wf-XML to the request shown in Figure 39
4.4 Vertical industry

The initiatives in vertical industry are to use XML to facilitate interoperability between their applications with industrial characteristics. Over 77 industrial projects are ongoing (Kotok 2000). For example, ACORD is for the insurance industry, groupDX.org for groupware, HL7 for healthcare, HR-XML for human resources, SAE for the automotive industry and REDX for retailing.

Some are overlapping, for example, financial industry has several projects: OFX (Open Financial eXchange), IFX (Interactive Financial eXchange), FinXML (Financial XML), FIX (Financial Information eXchange Protocol), FSTC (Financial Services Technology Consortium), and IOTP (Internet Open Trade Protocol). Although they have different focuses, much similarity exists, e.g., basic vocabulary and business processes. The thesis does not focus on this type, and here simply addresses OFX.

4.4.1 OFX (Open Financial eXchange)

OFX, created by CheckFree, Intuit and Microsoft in early 1997, is an industrial framework for exchanging financial data and instructions between customers and their financial institutions. It allows institutions to connect directly to their customers without requiring an intermediary.

From the perspective of technology, none is novel, because it:

- Uses a request and response model.
- Defines message structure using DTD.
- Transports messages via HTTP.

From the perspective of usage, it widens the applications of XML technology in financial interoperation. OFX offers a template to let financial institutions choose their vocabulary (a huge amount) and conventional requirements. OFX 2.0 offers several options for ensuring the security of customer transactions (OFX 2000). It also defines a large amount of vocabulary on aspects of:

- International support: language, currency and country-specific element values.
- Data synchronization: error recovery, use of multiple data files, and restoring from an outdated backup file.
- Financial profile: list of services and any general attributes of those services, and signon realms (user ID and password).
- Activation & account information: enrollment, accounts and activation.
- Customer to financial institutions communication.
- Recurring transactions.
- Banking.
- Payments.
• Investments.
• Bill presentment.

4.5 Specific applications

There are three reasons why we choose VoiceXML framework: (1) to broaden the application range of XML; (2) to increase the understandings of XML’s roles in integration; and (3) it is becoming more and more important.

4.5.1 VoiceXML

VoiceXML enables interactions with Web server through voice as well as the computer keyboard. VoiceXML defines dialogs between humans and machines in terms of audio files to be played, text to speech, and speech recognition capabilities, and touch-tone input. Its major goal is to bring the advantages of Web-based development and content delivery to interactive voice response (IVR)\(^\text{13}\) applications (VoiceXML 2000).

The architecture of VoiceXML is shown in Figure 41. A Document Server processes requests from a client application, the VoiceXML Interpreter or Voice Browser (Houlding 2001), through the VoiceXML Interpreter Context or a voice portal (Houlding 2001). The server produces VoiceXML documents in reply, which are processed by the VoiceXML interpreter. The Implementation Platform is controlled by the VoiceXML interpreter context and by the VoiceXML interpreter. For instance, in an IVR application, the VoiceXML interpreter context may be responsible for detecting an incoming call, acquiring the initial VoiceXML document, and answering the call, while the VoiceXML interpreter conducts the dialog after the answer. The Implementation platform generates events in response to user actions and system events. These events are acted upon by the VoiceXML interpreter or VoiceXML Interpreter Context.

A session of VoiceXML begins when the user starts to interact with a VoiceXML interpreter context, continues as documents are loaded and processed, and ends when requested by the user, a document or the interpreter context.

VoiceXML defines all the elements of an IVR transaction, including the grammars for each prompt, the recorded events, the selection of acceptable replies, and then the result

\(^{13}\) IVR is a term that is used to describe systems that provide information in the form of recorded messages over telephone lines in response to user input in the form of spoken words or, more commonly, dual tone multi frequency (DTMF) signaling. For example, when a user makes a call with a debit card, an IVR application is used to prompt the caller to enter a specific type of information, such as a PIN. After playing the voice prompt, the IVR application collects the predetermined number of touch tones (digit collection), forwards the collected digits to a server for storage and retrieval, and then places the call to the destination phone or system. Call records can be kept and a variety of accounting functions performed. In essence, the IVR application (or script) is a voice application designed to handle calls on a voice gateway, which is a router that is equipped with voice features and capabilities.
VoiceXML provides a form-filling mechanism for handling “normal” user input. Also, VoiceXML defines a mechanism for handling events not covered by the form mechanism.

In a common use of VoiceXML, a caller dials a telephone number that is routed to an IVR platform with a VoiceXML client, as shown in Figure 42. The platform translates the telephone number to a URL, and the client places an HTTP request to the specified URL. The Web server responds with a VoiceXML document that contains a dialog to be conducted with the caller. The client interprets the document to interact with the caller, plays prompts, collects input, and eventually submits the collected information to the URL designated by the document itself. The Web server processes the input and responds with a subsequent document to continue the session. (Danielsen 2000).

![VoiceXML Architecture Model](VoiceXML 2000)

**Figure 41** VoiceXML Architecture Model (VoiceXML 2000)
4.6 Syntax and Semantics

4.6.1 WSDL (Web Services Description Language)

WSDL, established by IBM and Microsoft, is the cornerstone of UDDI, which allows businesses to describe their offerings in a standard way. WSDL is an XML format for describing network services as a set of endpoints operating on messages containing either document-oriented or procedure-oriented information (WSDL 2000). The operations and messages are described abstractly, and then bound to a concrete network protocol and message format to define an endpoint. Related concrete endpoints are combined into abstract endpoints (services).

The services in WSDL are defined using six major elements (WSDL 2000):

- **types**, which provide data type definitions used to describe the message exchanges.
- **message**, which represents an abstract definition of the data being transmitted. A message consists of logical parts, each of which is associated with a definition within some type system.
- **portType**, which is a set of abstract operations. Each operation refers to an input message and / or output messages (including fault response). These operations can be categorized as four types:
  - **One-way**. The service receives a message.
• **Request-response.** The service receives a message, and sends a correlated message.

• **Solicit-response.** The service sends a message, and receives a correlated message.

• **Notification.** The service sends a message.

• **binding,** which specifies *concrete* protocol and data format specifications for the operations and messages defined by a particular *portType.* It allows *WSDL* in conjunction with SOAP, HTTP GET/POST, and MIME.

• **service,** which is used to aggregate a set of related ports.

• **port,** which specifies an address for a binding, thus defining a single communication endpoint.

We take the example in Figure 2 to explain *WSDL.* It defines one service, *StockQuote,* to query the last trading price of one stock. This service includes one operation, *GetLastTradePrice,* which is deployed via a SOAP protocol that uses the transport mechanism of HTTP. This operation consists of one input, *tickerSymbol,* and one output, *price.*

**types** in Figure 2 defines two elements, *LastTradePriceRequest* and *LastTradePriceResult,* using an XML Schema instance (xmlns = "http://www.w3.org/1999/XMLSchema"). The schema is used to define two messages, *GetLastTradePriceRequest* and *GetLastTradePriceResponse.* An abstract operation, *StockQuotePortType,* is defined via two formerly defined messages. So far the abstract definition of the service is over.

**binding** is used to define the concrete operation, *GetLastTradePrice,* implemented via SOAP over HTTP. It includes how to run the operation via *soapAction,* input and output with encoding style for the receiving applications to deserialize the SOAP messages. Finally **service** defines a set of addresses to represent communication endpoints using **port.** Here the service, *StockQuoteService,* just has one endpoint, *StockQuotePort,* which is available for business partners to run the operation.

Figure 3 shows how to run the operation. The request is the SOAP action http://example.com/GetLastTradePrice to query the last trading price of *ERICY,* and the response is a float, 20.5.

Figure 43 illustrates an informal syntax to describe the XML grammar of a *WSDL* document. Here it is important to understand

• the meaning of some characters: "?" (0 or 1), "*" (0 or more), and "+" (1 or more).

• *WSDL* is extensible through defining **extensibility element** shown in Figure 43. These elements are defined from external namespaces.
Figure 43  Informal syntax to define services using WSDL (WSDL 2000)
4.6.2 RDF (Resource Description Framework)

RDF is an XML application to describe metadata\textsuperscript{14} of Web resources to enable automated processing of them. RDF can be used in a variety of application areas, for example: in resource discovery to provide better search engine capabilities, in cataloging for describing the content and content relationships available at a particular Web site, page, or digital library, by intelligent software agents to facilitate knowledge sharing and exchange.

RDF describes metadata using a 3-tuple data model like resource-attribute-value (see Figure 44). The model shows three same relationships:

- **Value** is the value of **Attribute** for **Resource**.
- **Resource** has an **Attribute** with a value of **Value**.
- The **Attribute** of **Resource** is **Value**.

RDF Schema specification lets developers define a particular vocabulary for RDF data (such as authorOf) and specify the kinds of object to which these attributes can be applied. In other words, the RDF schema mechanism provides a basic type system for RDF models. This type system uses some predefined terms, such as Class, subPropertyOf, and subClassOf, for application-specific schema. RDF can be used to create semantic Web (Decker \textit{et al}, 2000; Berners-Lee \textit{et al} 2001).

The RDF and WSDL are general-special relationship. They have potential of collaboration to help each other.

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{figure44}
\caption{Resource Description Framework data model: three-tuples}
\end{figure}

\textsuperscript{14} In the context of Web pages on the Internet, the term metadata usually refers to an invisible file attached to a Web page that facilitates collection of information by automatic indexers; the file is invisible in the sense that it has no effect on the visual appearance of the page when viewed using a standard Web browser (Kobayashi and Takeda 2000).
4.7 Vocabulary

4.7.1 xCBL (XML Common Business Library)

xCBL was initiated by Commerce One, partner of eCo Framework. It realizes that each new specification starts from scratch, reinventing concepts (Commerce One 2000). These concepts may be general such as:

- Business description primitives like companies, services, and products;
- Standard measurements, date and time, location, country codes, currencies, business classification codes;
- Basic business forms like catalog, purchase order and invoice.

xCBL proposes an approach to developing XML document models from reusable semantic components. Such documents can be understood from their common message elements, while also providing a mechanism for linking to unique elements that vendors need to differentiate themselves. xCBL is to provide an initial set of XML building blocks that companies can assemble and extend to develop XML applications quickly. xCBL offers a path for companies with EDI implementations to migrate to XML-based e-commerce. All XML documents are defined according to XML DTD, SOX (Schema for Object-oriented XML) and XDR (XML Data Reduced).

Note that xCBL can be put in Specific Business Operations (Figure 8), where cXML is residing, because xCBL also provides XML-based formats for e-procurement. The reason we categorize xCBL in Vocabulary is to emphasize its role of providing basic XML components. Right now I have not seen any framework focusing on basic vocabulary for Internet commerce except xCBL.
5 An implementation of B2B e-procurement

5.1 Motivation

This project has multiple purposes, namely to:

1. **Explore the feasibility of open source software to develop B2B systems.**
   Open source software is growing quickly, so it is a good opportunity to try using open sources in e-commerce development. In the beginning, partially using open sources is set as a goal, but with the progress of development, we try to seek a way to use open source in the entire e-procurement system.

2. **Identify a B2B project for e-commerce education.**
   Find a suitable project is not an easy thing in e-commerce training. The project needs to (1) be interesting; (2) be challenging but not too difficult; (3) be cheap to implement; (4) be subject to a fixed time period; (5) gain sufficient experience and knowledge in both IT and business management; and (6) be demonstrated on a single machine. Implementing this project is beneficial to test these requirements.

3. **Learn the technology of XML and Java and feel its complementary role.**
   XML and Java are two fast developing areas. It is a popular idea to combine Java and XML in B2B software applications (Sundsted 1999; Johnston 2000), especially on the server-side (Maruyama *et al.*, 1999, p26). An attempt to use both leading-edge technologies to develop B2B software is of great importance. I hope to improve my skills in XML and Java through this project.

4. **Apply the auction to improve dynamics of pricing.**
   We assume that dynamic pricing via Auction is important in Internet commerce. Many Internet retailers apply this technology in their B2C portals. This implementation intends to use the technology into B2B e-procurement to realize direct procurement that enhances the agility of businesses. This style of purchasing will create dynamic pricing, meaning that the prices depend on the auction process and not fixed according to a contract signed beforehand.
5.2 Requirements capture

5.2.1 Inputs and outputs

To design an experiment to meet the motivations in 5.1, we suppose that a car manufacturer under the make-to-order environment uses the system to purchase some standard parts or components shown in BOM (bills of materials).

For the car manufacturer, the inputs include:
- The car orders placed by customers via any browser.
- BOM used to transfer car orders into part orders.
- Supplier information for the auction.
- Threshold data (if any part order is bigger than the threshold, the manufacturing warns to the subscribed suppliers in advance).

The outputs includes:
- Auction results (who win the part orders).
- Notifications to those order winners via email.

For the suppliers and ASP (Application Service Provider), inputs include:
- Inventory data.
- Price policy (different prices for different levels of order amount).

The outputs include receiving email to confirm the part orders if they are order winners.

5.2.2 Assumptions

We made the following assumptions for two reasons: to simplify the system and to focus on business-to-business interaction.

- For logistics, we do not consider how to transport the parts and components and fulfill the orders.
- For the auction, it is price-focused. A specific group of suppliers\(^\text{15}\) bid to one manufacturer and the lowest bidder wins. If the lowest price is bid by more than one supplier, the earliest one wins.
- For order fulfillment, one part order on a specified date is fulfilled by only one supplier. We exclude the possibility of a part order separately and partially fulfilled.
- For security, we ignore the security in e-commerce processes although it is extremely important. It is beyond the scope of this thesis.

\(^{15}\) We do not use open auction to let all possible suppliers bid because we are concerned about the qualification of suppliers. The qualification involves such factors as the product quality and delivery reliability.
We do not add the functions of manual entry while automating the whole business processes of B2B e-procurement.

5.2.3 Requirements

To demonstrate B2B procurement, the key requirement is how to interact between manufacturing server and supplier servers (See Figure 45):

1. At any specific time, the manufacturing server sends a part order to call supplier servers to bid.
2. Supplier servers process the part order.
3. Supplier servers respond via a new document with price information.
4. The manufacturing server processes the new document and checks if all part orders have been bid. If yes, stop; else, call an ASP (Application Services Provider) to fulfill the rest order.
5. The ASP server processes the rest order (supposing that the ASP is capable of fulfilling any order but with high price).
6. The ASP server sends a new document with price information to manufacturing server.
7. The manufacturing server processes the order and then stop.

To increase the ability of demonstrating the system, we add other requirements of B2C.

1. Let any customer place his car order using a Web browser.
2. The system gives the customer a thank-you response.
3. The system automatically transforms the car order into part orders.
4. Check if any part orders are bigger than their threshold. If yes, the manufacturing server sends a warning message to subscribed suppliers.
Figure 45  The architecture of the system
5.3 Business use cases

This section covers use case modeling to specify the e-procurement processes. First we define the system boundary. Because we want to use one Sun Solaris machine to show business-to-business procurement, manufacturing server and suppliers’ server and their interactions are inside the system. Outside the system is the customer who places the order to buy the cars he or she wants.

Besides the customer, we have to identify one event that invokes procurement processes automatically, here we choose Timer, meaning at some time the system initiates the process. In short, the system consists of two roles (see Figure 4616):

- Customer initiates B2C processes. Two business use cases are related to the role:
  - Convert the order into part orders.
  - Warn the suppliers.
- Timer triggers the B2B e-procurement processes. Three business use cases are involved in the role:
  - Procure by auction.
  - Inform the suppliers.
  - Fulfill non-bid part orders.

Three business workers are the Mfg (manufacturing) server representing the manufacturing server, supplier server, and ASP server. The four business use case realizations are Topic in JMS, Socket/ServerSocket, Javamail and SOAP, representing the technological implementation of five business use cases. These four approaches represent four modes of server-to-server communications.

---

16 The object-oriented analysis and design (OOAD) is implemented through Rational Rose 2000, e-development suite. This use case modeling and other diagrams seen later on are the results of ‘copy and paste’ from it. Note that Rational Rose is neither freeware nor open source.
Figure 46  Business use case diagram of B2B e-procurement
5.3.1 The role of Customer and their use cases

Customer is any user using a browser to fill the form on the Web to buy one or more cars. As the customer submits the buying form, he or she will receive a response to confirm the order. Figure 47 illustrates the basic flow of the events for each use case.

Convert the order into part orders

1. **Start of the use case.** The use case begins when the system receives the order placed by a customer.
2. **Verify the order.** The system checks if the order is valid on aspects of type and amount.
3. **Respond to the customer.** The system confirms the order and thanks the customer for placing the order here.
4. **Record the order.** The system records the order placement: date, name, customerId, email, telephone, and all items he or she wants to buy. The goal of doing this is for transportation later on or statistic analysis of customer information. The ideal storage is a relational database, but here I use Orders.xml instead, which is a text file.
5. **Read the date from bills of materials (BOM).** Synchronization with (4). To convert the order into a part order, we should read BOM information residing in BOM.xml, normally managed by the R&D department.
6. **Compute the part orders.** According to (3) and (4), it computes which parts and how many parts the customer order requires.
7. **End of the use case.** The system writes the part order information into PartOrders.xml. The tester can use a browser to validate its correctness.

Warn the suppliers

1. **Start of the use case.** It begins after the system converts the order into part orders.
2. **Compare the amount of part orders with the thresholds.** The system makes the comparison so as to make decisions whether to warn the subscribed suppliers to make compensations in advance.
3. **Publish the warning data in a topic server: JMS (Java message service).** Provided any part order is higher than the threshold, the system publishes them in a topic server defined in JMS. Otherwise, the alternative flow is the end of the use case.
4. **Receive the warning info.** The supplier servers receive the warning information, which can be defined as an event to trigger the backend systems of the supplier.
5. **End of the use case.** The system prepares to process the next order.
Timer is a trigger of B2B procurement from manufacturer to suppliers and back to the manufacturer. The business of manufacturing is assumed to start procurement at some time any day you want, for instance, this system selects 0:0:0, the very beginning of a day. Using time in B2B e-commerce is a natural action. In a limited market time, it is good because most activities occur during office hours. But in a global market, time has very little meaning. In this case it might be better to be event-driven. Figure 48 shows the activity diagram for the role of Timer.
Figure 48  Activity diagram for the role of Timer
Procure the part by auction

(1) **Start of the use case.** The system begins when the time reaches 0:0:0 every day.

(2) **Create an XML document from PartOrders.xml for bidding.** The system deeply imports the last child element of the top element of PartOrders.xml to a XML document. Here “deeply” means to copy the element and its all child nodes (attributes, elements and text nodes).

(3) **Send the document to the URL and Port number to call for bidding.** The system sends the document of (2) to URL and Port of (3). All suppliers’ URLs and Ports are saved in Suppliers.xml. The system parses it and reads one URL and Port number.

(4) **The supplier server receives the document.** The server receives the document and validates it.

(5) **The supplier server checks each part number in its inventory.** The server checks if there is inventory enough to fulfill this order. **Alternative:** if yes, go to (7); else, add an element: `<price>1000</price>`, then go to (6). We suppose that 1000 is a sufficiently high price so that we reject the bid.

(6) **The supplier server writes the price into the return document.** The system reads the price from PricePolicy.xml, and adds an element of `<price/></price>`. If the part is the last one in the document of (5), go to (8), else go to (6).

(7) **The supplier server sends the return document back to the manufacturing server.**

(8) **The manufacturing server receives the document.**

(9) **The manufacturing server writes the document information into Auction.xml.**

(10) **Alternative.** If the supplier is the last one of (3), go to (12), else go to (3).

(11) **End of the use case.** All bidding results are recorded in Auction.xml.

Inform the suppliers

(1) **Start of the use case.** The use case begins after the use case of Procure by auction is finished.

(2) **Create the mail body for each supplier.** The system reads one supplier, and collects all items it has won during bidding into one XML file, and adds the total amount and price the manufacturer will pay for it.

(3) **Send the mail body into its email server.** The system use Javamail to send the mail body to the supplier’s email address.

(4) **Supplier servers receive the bidding results.**

(5) **Alternative.** If this is the last supplier that won the orders, go to (5), else go to (2).

(6) **End of the use case.** The email send a message to inform the suppliers who won the orders.
Fulfill non-bided part order

(1) **Start of the use case.** The use case begins after the use case of Inform the suppliers is finished.

(2) **Alternative.** If any part bid with the price of 1000 in Auction.xml, go to (3), else go to (4).

(3) **Mfg server sends non-bid orders to ASP.** SOAP (see Section 2) is used to realize the function, thus all exceptions or faults are managed by the SOAP services.

(4) **ASP server receives the orders.**

(5) **The server decreases its warehouse amount.** We assume that this ASP has all products.

(6) **Write the price info to a new document.** The ASP server appends its price to the document originally created by the mfg server. This modified document is the new one.

(7) **Return the new doc.** The ASP service returns the top element back to the mfg server.

(8) **The mfg server receives the new doc.**

(9) **Modify the auction.** The mfg server adds the tag of price into Auction.xml so that all part orders are fulfilled.

(10) **End of the use case.**
5.4 Analysis and design

5.4.1 Class diagram for the whole system

Figure 49 describes all classes and their relationships. The exception is that only one supplier server exists in the diagram instead of four as in my demonstration system. To make the diagram easy to read, each class has only its name displayed and we omit the attributes and functions.

There are four packages in the diagram: HTopic, Tomcat Server, MySQL and SupplierEmailServer. We can find four kinds of dependencies to communicate between servers.

1. **Warn:** Pub/Sub, SupplierServer subscribes HTopic and HTopic publishes information to it.
2. **Call:** Socket/ServerSocket to achieve traditional client/server for bidding.
3. **Inform:** Javamail, manufacturing server to inform suppliers by email.
4. **Call:** SOAP, to call ASP to fulfill those non-bid part orders.

All XML documents start with DOM (Document Object Model). Each XML document is defined by a DTD (Document Type Definition). Figure 50 lists Auction.dtd.

Finally the whole system can be deployed in six components (See Figure 51): one manufacturing server (MidServer), four supplier servers and one ASP server.

---

17 The Document Object Model is a platform- and language-neutral interface that will allow programs and scripts to dynamically access and update the content, structure and style of documents. The document can be further processed and the results of that processing can be incorporated back into the presented page. There is an overview of DOM-related materials at W3C and around the web. For more information refer to www.w3.org/DOM/.
Figure 49  Class diagram of the whole system
The file of Auction.dtd

<!------Auction.dtd------>
<?xml version="1.0" encoding="UTF-8"?>
<ELEMENT Auction (Parts)+>
<!ELEMENT Parts (totalBid, Part+)>
<!ATTLIST Parts froWhichDay CDATA #REQUIRED>
<!ELEMENT totalBid (#PCDATA)>
<!ELEMENT Part (validBid, Choice+)>
<!ATTLIST Part name CDATA #REQUIRED>
<!ATTLIST Part number CDATA #REQUIRED>
<!ELEMENT validBid (#PCDATA)>
<!ELEMENT Choice (price, supplier, supplierEmail)>
<!ATTLIST Choice time CDATA #REQUIRED>
<!ELEMENT price (#PCDATA)>
<!ELEMENT supplier (#PCDATA)>
<!ELEMENT supplierEmail (#PCDATA)>

Figure 50  The file of Auction.dtd
5.4.2 Detailed Classes

For reading convenience, Figure 49 does not carry all attributes and functions in each class. This section gives a detailed specification of one complicated class: ThreadForMidServer (See Figure 52).

One function in the class of AuctionProcess is process, which is to process the XML document bid by each supplier. The concrete algorithm is illustrated in Figure 53.
Figure 52  Detailed class diagram of ThreadForMidServer
Figure 53  Activity diagram of Process in ThreadForMidServer
5.5 Implementation Processes using Java and XML

5.5.1 Identifying important components

Six important components are used in this experiment, that is, Java servers, XML parser or processor, relational database that supports JDBC, SOAP implementation, Java mail implementation, and JMS implementation. I choose these components based on the following general criteria:

- **Open sources.** You can read source codes and change and drop them without any cost. This is useful, so I attempt to make it as one of features of this experiment.

- **Without time limitation.** Most application servers allow use within one or two months; beyond this time range you have to pay expensive license fees. For example, I have reviewed IBM’s server, WebSphere standard version, and Inprise/Borland’s server, IAS, although both had better features and are more suitable for this project, I did not choose them.

- **Small disc amount needed.** Because I run the whole process of B2B procurement on one machine, I need to compute the disk requirements and my disk capacity. To reduce the difficulty, I have implemented a simple server used by all suppliers to simulate their behaviors.

- **Upgrade quickly to meet new specifications.** This is important for XML programming because XML-related specifications are not stable and are changing rapidly such as XML Schema, DOM2 and SAX2. If the software cannot change with the pace of the specifications, you cannot use the cutting-edge technologies in a project.

- **Easy to learn.** They offer good examples to let you start up and mail lists to solve tough programming issues.

Now we will see how to choose the components in this project (see Table 5):

**Java servers**

Java servers are used to support two Java packages: javax.servlet and javax.servlet.jsp, to process the orders that the customers place via Web browsers. I chose JSWDK (JavaServer Web Development Kit) from Sun Microsystems because it fits all general criteria specified above and also it provides one good example for online buying, which is later on configured as the front page of Zhao’s Car shop. JSWDK 1.0 supports Java Servlet 2.1 and JSP 1.0. The JSWDK server is simulated as an MfgServer in this project.

As I find JSWDK was donated to Apache as Tomcat (http://jarkata.apache.org), I downloaded Tomcat 3.1, which supports Java Servlet 2.2 and Java Server Page (JSP) 1.1. This server is used to deploy my SOAP applications.
XML parsers
Most XML parsers are freeware. I have used Sun’s JAXP 1.0 (Java API for XML Parsing), and IBM’s XML4J. However, as I found open source software Xerces-J in Apache (xml.apache.org/xerces-j), I have been using Xerces-J (V 1.03-1.30). Xerces-J is easy to use. My XML programming skills have been learned from its mail lists.

Relational database
In the processes of e-procurement, there are many persistent data to store and reuse. I use mySQL database, an available module on my local machine. Moreover, it offers a free JDBC driver to link its database. Tutorial materials are also easy to get from the Internet http://www.mysql.com/downloads/. Note that mySQL is freeware not open source software.

JMS implementation
JMS provides a standard Java-based interface to the messaging services of a MOM (Message-oriented middleware) or some other provider (Asbury and Weiner 1999).

Java mail package
I use this package from my local machine, implemented by Sun Microsystems. You can download it (j2ee.jar) from http://java.sun.com/j2ee/, because Sun released it together with J2EE (Java 2 Enterprise Edition).

SOAP implementation
I downloaded it from Apache-Soap: http://xml.apache.org/soap/, donated by IBM.

Table 5  Open source or freeware used in this experiment

<table>
<thead>
<tr>
<th>Name</th>
<th>from</th>
</tr>
</thead>
<tbody>
<tr>
<td>Java Web servers</td>
<td>Apache’s tomcat: <a href="http://jakarta.apache.org/tomcat/index.html">http://jakarta.apache.org/tomcat/index.html</a></td>
</tr>
<tr>
<td></td>
<td>Sun’s JSWTK: <a href="http://java.sun.com/products/servlet/archive.html">http://java.sun.com/products/servlet/archive.html</a></td>
</tr>
<tr>
<td>Relational DB</td>
<td>Freeware</td>
</tr>
<tr>
<td></td>
<td>MySQL: <a href="http://www.mysql.com/downloads/">http://www.mysql.com/downloads/</a></td>
</tr>
<tr>
<td></td>
<td>JDBC driver: same as above</td>
</tr>
<tr>
<td>JMS interface</td>
<td>Open source</td>
</tr>
<tr>
<td></td>
<td>Sun’s J2EE: <a href="http://java.sun.com/j2ee/">http://java.sun.com/j2ee/</a></td>
</tr>
<tr>
<td>JMS implementation</td>
<td>Open source</td>
</tr>
<tr>
<td></td>
<td>miniJMS (Asbury and Weiner 1999) or sonicMQ: <a href="http://www.progress.com/sonicmq/">http://www.progress.com/sonicmq/</a></td>
</tr>
<tr>
<td>JavaMail package</td>
<td>Open source</td>
</tr>
<tr>
<td></td>
<td>Sun’s J2EE: <a href="http://java.sun.com/j2ee/">http://java.sun.com/j2ee/</a></td>
</tr>
<tr>
<td>SOAP implementation</td>
<td>Open source</td>
</tr>
<tr>
<td></td>
<td>Apache-SOAP: <a href="http://xml.apache.org/soap/">http://xml.apache.org/soap/</a></td>
</tr>
</tbody>
</table>
5.5.2 Practical hints

This section describes how to initiate each module by setting development environments.

- Java Web servers. Add appropriate servlet.jar to classpath. Here “appropriate” means its version fits the requirements of the servers. For example, JSWDK 1.0 supports Java Servlet 2.1 and JSP 1.0. Then you can change the server’s port number by editing the XML file. Finally you can start up or shut down your server at the port number you choose, referring to its Web site.

- XML parsers. Add xerces.jar to your classpath.

- Relational Database, you must start MySQL server and add the JDBC driver (mm.mysql.jdbc-1.2c) to your classpath.

- JMS interface and Javamail package: Add j2ee.jar to your classpath.

- JMS implementation. Because JMS is implemented via RMI, you must start rmi server by rmiregistry portNumber in Sun Solaris. Then change the host name into my host name (mir20.ida.liu.se). Finally, add minijms.jar to your classpath.
  - To start queue (point-to-point), use java HHQueue mir20.ida.liu.se.
  - To start topic (publish/subscribe), use java HHTopic mir20.ida.liu.se.

- SOAP implementation. Add soap.jar to your classpath.
5.6 Testing

Bottom-up testing is used in this project. It consists of unit testing (all Java classes and packages), use case testing and system testing.

5.6.1 Unit testing

Unit testing is to test all Java classes and components. To test each Java class, we use the Java compiler from Sun Microsystems in accordance with the requirements in Figure 46. To test if each Java class is what we want, we can use the “return identifier” to see the result externally, and extensively use “System.out.println ()” to see its internal runs.

Deployed components include:

- Java Web server: JSWDK 1.0 and Tomcat
- HHTopic in JMS
- MySQL database and its JDBC driver

Testing such a component requires designing a class having main method to use its API. For example, MySQL has its SQL interface and JDBC driver, and HTopic in JMS has its Publish/Subscribe interface, and Java servers have their startup.sh and shutdown.sh script files. In addition, Java servers have to run for a period of time, i.e., two weeks, to observe their robustness.

5.6.2 Use case testing

This section describes how to test five use cases identified in Figure 46. The inputs and outputs of each use case are demonstrated in Table 6. Two things should be stressed:

- Microsoft’s Web Browser, IE 5.0, offers a default style sheet to read XML documents. Thus it is convenient to read them using IE 5.0.
- We can change the parameter of input factors to see the outputs under different conditions. For instance, for the use case of Warning to the suppliers, we can first change the number in Threshold.xml one by one, then change some numbers group by group. For each change, we can input different amounts of orders placed by Customer. Then we see if the warning information is as we expect.
Table 6  Use case testing

<table>
<thead>
<tr>
<th>Role</th>
<th>Use Case</th>
<th>Inputs</th>
<th>Outputs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Customer</td>
<td>Convert the order into part orders</td>
<td>The order placed by any customer synchronically; BOM.xml</td>
<td>Orders.xml; PartOrders.xml</td>
</tr>
<tr>
<td></td>
<td>Warning to the supplier</td>
<td>The amount of the order placed by any customer. Threshold.xml</td>
<td>Publish info in HTopic; Received information by four Supplier Servers</td>
</tr>
<tr>
<td>Timer</td>
<td>Procure by auction</td>
<td>PartOrders.xml; Process algorithm; Supplier.xml; Inventory.xml &amp; PricePolicy.xml in four supplier servers</td>
<td>Auction.xml in manufacturing server;</td>
</tr>
<tr>
<td></td>
<td>Inform the suppliers</td>
<td>Auction.xml; Supplier.xml</td>
<td>Emails received by four supplier servers</td>
</tr>
<tr>
<td></td>
<td>Fulfill non-bid part orders</td>
<td>Auction.xml; ASP server information</td>
<td>Warehouse and Price in MySQL; Auction.xml</td>
</tr>
</tbody>
</table>

5.6.3  System testing

System testing is to test the whole system behavior. To realize the goal, this project starts with setting up the parameters in the system, and then runs all the servers (manufacturing server, 4 supplier servers, HTTopic server, SOAP server), and finally lets the system run for two weeks. During the period, we can place the orders from different browsers on different platforms, and observe the results of Auction.xml, and check the emails of suppliers and warehouse information of ASP.

5.6.3.1  A typical test case

Figure 54 lists one auction result in Auction.xml (Auction.dtd shown in Figure 50). The results are tagged with <Parts> and sorted according to date. We only record the auction result on Feb 21, 2001. We need to buy five parts, and each keeps two choices of suppliers. In fact we can make any number of choices in this implementation through setting maxChoice in class ThreadForMidServer.

The document shown in Figure 54 has two special tags: totalBid and validBid. The former is to record the auction time whereas the latter shows the valid bid time. The supplier will bid a sufficient large price (e.g., 1000), which is defined by the auction community, if the bidding is not valid as the supplier cannot fulfill this orders in case it has no enough inventory.

We use Javamail to inform those suppliers who win the orders. One supplier will receive at most one email no matter how many orders he wins. In this case (Figure 54), the body of emails is as shown in Figures 55-58. Supplier9916 wins two part orders
(tire501 and brake) in Figure 55, Supplier9926 wins one part order (tire502) in Figure 56, Supplier9936 wins one part order (engine) in Figure 57, and the ASP wins one order (bearing) in Figure 58. The email body includes supplier name, supplier email address, total number, and total payment. The real situation can be as complex as an invoice.

5.6.3.2 Expert evaluation

To evaluate the system, we invited an expert working as a B2B consultant at a leading e-business software vendor. Some points are as follows:

- His company prototypes such kinds of systems.
- Warning to suppliers in advance is of interest. He has not seen such systems.
- “Focus on price” in the auction is the right customer requirement.
- It is important that the supplier’s links to the participants in the auction are not open. Some factors such as product quality and delivery are also important. Only those who meet the basic requirements are qualified to bid. Open auctions may imply big risks.
- Overemphasizing automation is sometimes dangerous, because the communication cost is high, the implementation is time-consuming, and keeping the key personnel that lead the automation implementation is hard. In most situations, it is important to parallel manual and automation.
- XML has been used in data communication between ERP and other systems.
- From the commercial standpoint, the system should consider more features, for example, more data items and combination with logistic processes.
<xml version="1.0" encoding="UTF-8" ?>
<Auction>
<Parts forWhichDay="20010221">
<totalBid>4</totalBid>
<Part name="tire501" number="63">
<validBid>3</validBid>
<Choice time="19:20:03">
<price>4</price>
<supplier>Supplier9916</supplier>
<supplierEmail>zhao_yuxiao@yahoo.com</supplierEmail>
</Choice>
<Choice time="19:20:06">
<price>4</price>
<supplier>Supplier9926</supplier>
<supplierEmail>zyx66@hotmail.com</supplierEmail>
</Choice>
</Part>
<Part name="tire502" number="4">
<validBid>4</validBid>
<Choice time="19:20:06">
<price>11</price>
<supplier>Supplier9926</supplier>
<supplierEmail>zyx66@hotmail.com</supplierEmail>
</Choice>
<Choice time="19:20:08">
<price>11</price>
<supplier>Supplier9936</supplier>
<supplierEmail>supplier3@eudoramail.com</supplierEmail>
</Choice>
</Part>
<Part name="engine" number="13">
<validBid>2</validBid>
<Choice time="19:20:08">
<price>0.4</price>
<supplier>Supplier9936</supplier>
<supplierEmail>supplier3@eudoramail.com</supplierEmail>
</Choice>
<Choice time="19:20:03">
<price>0.44</price>
<supplier>Supplier9916</supplier>
<supplierEmail>zhao_yuxiao@yahoo.com</supplierEmail>
</Choice>
</Part>
<Part name="brake" number="14">
<--similar to "tire501", ignore some elements-->
</Part>
<Part name="bearing" number="70">
<validBid>0</validBid>
<Choice time="19:20:13">
<price>40.0</price>
<supplier>Application Service Provider</supplier>
<supplierEmail>asp18z@netscape.net</supplierEmail>
</Choice>
</Part>
</Parts>
</Auction>

Figure 54  Auction results on Feb 21, 20001 (Auction.xml)
Figure 55  Supplier9916 wins two part orders (tire501 and brake)

Figure 56  Supplier9926 wins one part order (tire502)

Figure 57  Supplier9936 wins one part order (engine)

Figure 58  ASP wins only one part order (bearing)
5.7 Discussion

Now we should return to Section 5.1 to answer the questions that initiated this project and address some unexpected issues in B2B development. First we emphasize that not all functions in B2B e-procurement are implemented in this project, e.g., payment.

5.7.1 Open source or freeware in developing B2B e-commerce

Section 5.5.1 describes the open source or freeware used in this project: Java Web servers, XML parsers, relational database, JMS interface, JMS implementation, Javamail package, SOAP interface and implementation. Although this project is not developed for enterprises, you can use these open sources in enterprise software development. To gain the enterprise characteristics such as scalability and robustness of Tomcat and JSWDK (Java Web Server), you can combine Tomcat with Apache Web server, Websphere, WebLogic or iPlanet.

I have not found any major difficulty using open source software. One practical issue is that I have to update my downloaded software monthly because they (Xerces-j and SOAP) are improving daily even though the version has not been updated. In general it is important to address how to search open source or freeware in a formal way so as to get more options to select the most appropriate one.

Compared to the drawbacks, there are many more benefits I gained.

• Free of charge.
• Download them anytime you want.
• Do not consider future use because I can use it as long as I want unlike most commercial software that you can only try a fixed period of time.
• Open source software has the potential to be accepted as standards so as to enhance the value of the applications based on open source. For example, Linux has such a trend (IBM 2000).
• Receive quick response if you send email to the mail list to inquire about any issue of installation, configuration, programming and maintenance.
• Read all the Java code so that you quickly become a Java expert because you can learn such things as Java application frameworks and design patterns.
• Learn how to change specifications to OOAD and Java code, from DOM specification to Xerces-j, and from SOAP specification to SOAP implementation.

As to the freeware, you cannot gain the last three benefits mentioned above. But you will find open source and freeware are reliable as well.

Then, one open question is whether we can use them in commercial use or license issue. Open source licenses regulate the following (Konstroffer et al 2000):

• Everybody is able to read, modify and redistribute the software.
• The license may not discriminate against anybody.
• Changes made to free software must be made available to the community.

For companies to use open source, the last one is restrictive. It means that every extension or addition made to the open source is also free. Sometimes it is difficult to use it.

For MySQL (freeware), it costs nothing for normal use. Only when you sell MySQL directly or as a part of another product do you have to pay for it. The current price is $200 for one license, and $1000 for 10 licenses\(^{18}\) (Klauser 2000). Exceptions to this are Microsoft operating systems. This is a shareware version of MySQL called MySQL-Win32, which may be used for evaluation purposes, but a usage license is required.

Contrasting the license of MySQL and that of open source, you can pay the license fee in MySQL to exchange not-free-of-your-own-extension right in open source.

As we completed the software implementation, a paper (Konstroffer \textit{et al}, 2000) validated the approach: open source software is a good way to develop e-commerce systems.

\subsection*{5.7.2 Designing a suitable project for e-commerce training}

We demonstrate that B2B e-procurement is a suitable project for students to learn e-commerce.

• It is interesting and leaves many open issues to let students create new ideas.
• It uses the leading-edge technology (XML and Java).
• The time range is about two months.
• You can choose any freeware or open sources and importantly you can find a number of alternatives.
• You can demonstrate the multiple-server communication on one machine.
• You can learn software processes from modeling to testing and some business skills (e.g., purchasing, BOM and dynamic pricing).

This project is defined to implement a B2B e-procurement using Java and XML. The time range can be two months but if the students are not familiar with Java and XML knowledge it can take longer. No commercial software is used. The evaluation is based on the documents of implementation process and software demonstrations.

\subsection*{5.7.3 XML and Java are complementary}

We use XML and Java to implement the whole system. XML and Java in B2B development are complementary. Although they are not sufficiently mature, both were originally developed towards the Internet, but address different issues.

---

\(^{18}\) What does this mean if you are running a commercial Web server using MySQL? In this case, because you are not directly selling MySQL, you are not required to purchase an MySQL license. However, the designers of MySQL would like you to purchase email MySQL support, which costs $200 a year (Klauser 2000)
XML is used to describe the information that communicates between applications and most importantly the applications can read its meanings because of its specific and extensible tags. DTD and XML schema can define any instances of an XML document you want. But XML itself is static and stateless unlike HTTP that defines the transport mechanism (TCP/IP) and packages (header and body), and thus XML requires being transported, recognized and processed.

Java as an Internet language offers a wide range of packages such as java.io, java.net to handle the messaging requirements. And also the free-of-charge Java parsers allow XML to be easily serialized and deserialized for network transport, and read and manipulated by applications. In addition, many new Java packages are emerging to broaden this advantage. We apply the Java packages of SOAP via HTTP, Javamail via SMTP, and JMS. You can find many commercial implementations using Java if you search on the Internet. Conversely you would not be so lucky if you searched for C or C++ implementation.

The unexpected issues arising from the use of XML and Java are as follows.

**Parsing XML documents using a DOM parser can produce memory problems.**

Using a DOM parser you can look through the whole XML document and conduct such functions as search for the whole document. The problem is that the whole XML must reside in memory. If it is large enough, it will create memory overflow. In Xerces2, the next generation parser of Apache-Xerces-J, the developers identified this requirement ranking it No 219 (Apache-Xerces 2000):

*The parser should have a competitive memory footprint (competitive against the other Java based parsers out there).*

We handle the large XML document by dividing the document into small ones.

**Writing a good DTD is not easy.**

Excluding naming problems, Figure 59 shows four alternatives to describe an item in a customer order: *brand is Volvo and number is 2*. The first is most easily conceived but it has three levels (Item—brand—”Volvo”) to express the meaning. The second has one level but the number can be changeable in the XML document manipulation. The third directly uses Volvo as the element name, which may make software reuse difficult. The last one seems good but we find cXML and xCBL did not select this style.

During my programming, I first avoided the third one because it affected my randomly changing the brand name. I also did not use the last one. Because the element of brand and number belongs to the same level concept in my mind, so both should lie on the same level. How did I select the rest two alternatives? I used the first one if the number was modified frequently, otherwise I used the second.

To achieve interoperability, taking a common one is necessary (Maruyama et al. 1999, p251). Here common means industry standards or recommendations by non-profit organizations. But there are also drawbacks to them (Bradley 2000, p77):

---

19 The no. 1 requirement is *the code shall be maintainable and simple to read.*
• The DTD is not perfectly customized, i.e., important information is not be tagged or be tagged inappropriately.
• The DTD may contain information that is never used because all industry standards aim at completeness.
• The DTD rules may be too flexible to meet various requirements of various enterprises, and thus too difficult to achieve interoperability.

Taking a suitable DTD, you have to make a compromise decision to balance such factors as interoperability, customization and reusability.

Reusing software that manipulates XML documents is a little difficult.

Suppose that the software knows the DTD of the XML document, it has to consider the writing style related to whitespace in markup. The writing style is not defined by DTDs, and you can use any text editor to edit the XML document in theory.

To avoid the problem above, you can use the method in org.w3c.dom.Element

GetElementsByTagName (java.lang.String elementName)

To create a NodeList, all descendent Elements with a given tag name are listed in the order in which they are encountered in a preorder traversal of this Element tree. The traversal will spend a lot of time, which can be critical in some real-time systems.

Another difficulty in reusing software involves XML parsers. If two applications use different parsers or different versions of the same parser, reuse can be difficult. The main reason is that the XML parser development is ahead of DOM standardization.

Suppose that the software knows the schema of the XML document, in the short term, the late adoption of XML schema by W3C also creates reuse problems. For instance, the year 2000 saw two major change of XML Schema in April and September.

![Four alternatives DTDs (brand and number in order)](image)

**Figure 59** Four alternatives DTDs (brand and number in order)
5.7.4 Dynamic pricing based on auction

We implemented dynamic pricing based on auction, which may be important in reducing cost and increasing agility, especially for standard parts and components. The auction can be one buyer and multiple suppliers, or one supplier and multiple buyers. And the auction can be open or closed to qualified players. Our case is one buyer and specific multiple suppliers.

5.7.4.1 Business processes

At least seven business processes are involved:
1. A community is established to include a buyer and multiple suppliers.
2. The buyer prepares the order list.
3. The buyer sends the list or the suppliers request the order list.
4. The suppliers process the order list based on their own business context.
5. The suppliers reply to the order list.
6. The buyer processes the order bidding information, according to some algorithm.
7. The buyer informs those suppliers who win the orders.

Our implementation is simplified as follows.
1. We use the document suppliers.xml to record all supplier information including Internet address and port number of supplier servers, and email address.
2. The buyer server creates an XML-based order from PartOrders.xml.
3. The buyer directly sends orders to suppliers one by one to let them bid.
4. The supplier servers process the bidding XML documents according to inventory information and price policy (different prices for different amounts of buying).
5. The supplier servers send the bidding order back to buyer server.
6. The buyer server selects the lowest price as winner of each item. If the price is same for more than one supplier, the earliest wins the order. If not all items in the customer order are bid, the server calls an ASP to fulfill the rest orders as we assume the ASP can offer any items but at a rather high price.
7. The buyer server uses Javamail to inform the suppliers and the ASP.

5.7.4.2 How to guarantee the rights of suppliers

In our implementation case, there are one buyer and multiple suppliers. The suppliers stand in a disadvantage position towards the buyer, because they must change quickly to meet the changing demands of the buyer, they do not have complete bidding information of other suppliers, and they must carefully serve the buyer at any time (long term and short term). It is the potential major orders of the buyer that maintain the long-term relationship of such a community. For the case of small orders, perhaps we should consider some conditions to guarantee the rights of suppliers. If we can define such conditions, auction-based dynamic pricing can be used in environments of small orders.
5.7.4.3 Not suitable for all products

We use auction pricing to buy car parts and components, which tend to be standard products. Obviously, auction pricing cannot be applied to some products. Innovative products are one example. Auctions require many suppliers, the more the better. But innovative products are often owned by very few players. Price is not a deterministic factor.

5.7.4.4 How to measure those not-easy-to-measure factors

The not-easy-to-measure factors include the product quality, reliability of order fulfillment, agility, delivery and innovation. Auction processing only considers price, but price is not the only competitive advantage. We need a standard system of classification and certification. Because of this difficulty, we predict that some non-profit organizations would start up some dynamic certification marketplaces for B2B marketplaces to certify or index these factors so as to choose the most suitable partners. These authoritative certification marketplaces will be welcomed by the customers and facilitate some legal issues between trading partners.

5.7.4.5 How to handle exceptions in automatic processes

Another problem is how to process exceptional situations to enhance the dynamics of the whole system and thus decrease such operational benefits as quick response. Here exceptional means those things that happen that the system did not expect and actually addresses the flexibility of business process mapping.

Handling the exceptions completely can be very complex considering the whole supply chain. For example, if we cannot buy the parts to build a car, when and how do we trigger an event to a suitable process in the supply chain to reprocess the order again, and how should we negotiate with customers or can we cancel the order? The ideal solution is to optimize the supply chain to find one best route. However, optimizing inter-enterprise supply chains in a highly distributed environment is difficult. Moreover, if some partners are willing to apply to join or leave the chain dynamically, which is a normal case in e-commerce, the problem can be much tougher.

5.7.5 XFIC and e-procurement implementation

We discuss 15 frameworks in Chapters 2-4, and this section discusses what frameworks can be principally used and what frameworks were used in this implementation.

What XFIC can be used to fulfill the requirements described in 5.2.3?

- SOAP: the Mfg server requests suppliers to bid; the Mfg server informs suppliers that won the part orders; the Mfg server calls ASP to fulfill the rest part orders. BizTalk cannot be used because our implementation is on Sun Solaris and I have not found any implementation of a BFC server for Sun Solaris.
- cXML: Request-response communication model and related document formats (Ariba 2000, p 46-57) can be applied for the Mfg server to request suppliers to bid and to the Mfg server to call ASP to fulfill the rest part orders. One-way
asynchronous model and related document formats (Ariba 2000, p57-61) can be applied to the Mfg server to inform those suppliers who won the orders.

• xCBL: Many XML formats can be used in this implementation, e.g. OrderRequest and OrderResponse; AuctionCreate, AuctionCreateResponse, AuctionResult, AuctionResultResponse and RequestForQuotation (CommerceOne 2000). Note that both xCBL and cXML could be embedded into SOAP so that the Mfg server requests and informs suppliers and calls ASP to fulfill the rest orders, referring to see Figure 26, which shows how BizTalk is implemented via SOAP.

• ICE: the Mfg server publishes warning information to suppliers; Supplier servers subscribe warning information to the Mfg server. It is suitable to use a 1-n syndicator/subscriber scenario (see Figure 30-33).

• RosettaNet: Order Management (PIP3) can be used to formalize the interaction between the Mfg server and the supplier servers, including QueryPrice&Availability (PIP3A2) and NotifyPurchaseOrderAcceptance (PIP3A7).

• WSDL: It can be used to describe the functionality of the Mfg server and supplier servers to present other business partners. If we want to register the services of the Mfg server or supplier servers in UDDI operators, this is the start.

What XFIC have been used in this implementation?

Only SOAP was used for the Mfg server to call ASP to fulfill the rest part orders. The main reasons are:

• SOAP is becoming industrially accepted.

• SOAP implementation in Java is available as open source software.

• Simplifying the programming to avoid using the message formats represents general situations.

• It is not important to implement the fast changing XFIC.
6 Summary and Conclusions

This thesis contains two major contributions. One involves XML-based frameworks for Internet commerce (XFIC). This thesis helps explain what they (XFIC) are and how they interact. We chose 15 most important frameworks (see Tables 1 and 2): ebXML, eCo Framework, UDDI, SOAP, BizTalk, WSDL, cXML, ICE, Open Applications Group, OFX, RDF, RosettaNet, Wf-XML and xCBL. The other involves an implementation of B2B e-procurement. In addition, this chapter also addresses how to apply XFIC in e-commerce development, and presents future research.

6.1 What XFIC are

- Definition. XFIC are templates used to automate XML messaging between applications in order to meet the requirements of e-commerce. The applications can reside internally or between business partners. The goal of XFIC is to achieve higher interoperability in heterogeneous environments. XFIC are the cornerstone to developing interoperable Web services.

- Philosophy. The philosophy behind XFIC is to make a compromise between extensibility and interoperability of XML. Using XFIC, you cannot arbitrarily extend XML documents but you can make applications communicate more easily.

- A variety of industrial initiatives for XFIC that cover different niches of e-commerce. The fifteen most important and active XFIC are described in Chapters 3 & 4.

6.2 How XFIC interact

A hierarchical model is shown in Figure 8 to demonstrate the different purposes of XFIC. We start with the generic model of XFIC (see Figure 4), where two applications communicate via XML messaging.

- As we go up, we assume that applications are not known beforehand, thus you need to find them through registry and search mechanism. We call this layer the big picture, including functions of Register, Search, Negotiate, Contract and Configure. Three frameworks partially take up these roles, i.e., ebXML, eCo Framework and UDDI. eCo Framework is the earliest initiative but has wrapped up in 1999. ebXML, active and widely supported, models e-business as collaborative processes and strives to fulfill all requirements of business collaboration, transaction, messaging, document definition, and choreography. UDDI focuses on registry only, and provides a quick solution to let participants register their Web services and access one another via the registry.
As we go down the model, we assume that applications are specified and known to each other; thus you need to know how to quickly implement XML messaging. Three layers are included aimed at messaging, syntax & semantics, and vocabulary.

- SOAP offers simple messaging using RPC, and BizTalk is an application of SOAP adding a BFC server for enterprise capability. Four specific messaging approaches are **specific business operations** (cXML for procurement, ICE for 1-n subscribe/publish), **EA1** (OAG for integrating internal enterprise components, RosettaNet for supply chain integration, and Wf-XML for workflow integration), **vertical industry** (OFX for financial industry), and **specific applications** (VoiceXML for voice-related applications).

- Syntax and Semantics are to make XML document understandable especially for computers. Syntax shows how to structure XML items and semantics describe the meanings of XML items. WSDL provides the grammar to define Web services and detailed operations. RDF is used to describe the metadata of Web contents.

- XML vocabulary is used to formalize the use of XML tags for broad interoperability. xCBL is created to offer common vocabulary in e-commerce such as customer order and address.

Cooperate on standards and compete on implementation has become a common sense for companies. A relationship model (Figure 9) is given to show the cooperative and competitive relationships between XFIC. The core competitive relationship is presented within two groups: ebXML and UDDI/SOAP/WSDL. The former is initiated by non-profit organizations (OASIS and UN/EDIFACT) and the latter represents industrial solutions (IBM, Microsoft, etc). However, a merger is unavoidable in the future. For example, ebXML has determined to integrate SOAP into its messaging service specifications.

Interoperable Web services development is becoming feasible. A chronological model (Figure 10) is provided to view XFIC from the perspective of evolution. September 2000 is a milestone to separate XFIC development into two stages: from proprietary solutions to open and interoperable Web services.

### 6.3 An implementation of B2B e-procurement

The implementation of B2B e-procurement demonstrates the following conclusions:

- It is possible to use open source or freeware in B2B e-commerce development. Our project finds using open source software much more benefits than disadvantages: free of charge, use it under loose conditions, easy support, and learning. License is a small problem, but you pay a much lower license fees to keep update in a correct way.

- We identify a B2B project for e-commerce training: implement a B2B e-procurement using Java and XML. The time range can be 2 months but if the students are not familiar with Java and XM it can take longer. No commercial
software is used. The evaluation is based on the implementation documents and software demonstrations.

- XML and Java are complementary: XML for contents and Java for automating the contents. But some problems exist:
  - Parsing an XML document using a DOM parser can produce memory problems.
  - Writing a good DTD is not easy.
  - Reusing software that manipulates XML documents is a little difficult.
- We implement auction-based dynamic pricing during procurement processes. We find that:
  - The business processes involved can be complex, thus it is hard to handle exceptions in the whole automatic processes.
  - Auction-based pricing is not suitable for all products, especially innovative ones.
  - How to guarantee the rights of suppliers is an open problem.
  - There are some not-easy-measure factors.

6.4 Applying XFIC in e-commerce development

I am often asked about how to apply XFIC in e-commerce development when I present my work to other people. This section focuses on some important issues related to the applications of XFIC, although previous chapters have addressed the applications of XFIC to some degree.

6.4.1 The benefits of using XFIC

The benefits of applying XFIC in e-commerce development can be summarized as follows:

- To find a formal and simple way to develop, deploy and register interoperable Web services for conducting e-business.
- To save a lot of time and energy in using the vocabulary, syntax and standards (open standards). We give one example to show that writing a DTD is not easy in 5.7.3. If you use the existing data formats written by XML professionals, it will certainly save you much work.
- To achieve higher interoperability with your customers and suppliers under some conditions.
- To leverage the extensibility of XML.

6.4.2 The barriers in applying XFIC

There are some technical barriers in applying XFIC in software development, which include:
1. XML standards are underdeveloped, for example, XML Schema could offer a variety of date types if it were standardized.

2. Nearly all frameworks are changing quickly. It is impossible to develop interoperable software based on unstable specifications.

3. The frameworks represent the general requirements, resulting in complex programming. For example, Figure 28 represents a general customer order of cXML, which includes all items a customer order demands such as payment and tax. If you select cXML completely, your programming will become very complex to handle all items although you are interested in only a part of the elements.

4. The applications of some frameworks have to subject to some important conditions. For example, SOAP has to combine with the servers that communicate with each other.

5. As for EDI users, they do not want to abandon a long-term investment in EDI because there is no solid return-on-investment of moving into XML (Meehan 2001).

6. Many frameworks coexist that create new integration issue. Compared the customer order in BizTalk (Figure 26), cXML (Figure 28) and OAG (Figure 34), they are rather different. It is hard to automate the processes if you and your partners use different message formats.

7. Security is involved. Not all frameworks directly address the issues. The common solution is that the transport mechanism is bound to HTTP and security is supported with certificate-based authentication via HTTP/SSL.

6.4.3 How to choose the right frameworks

First of all, it is very important to choose the right XFIC, because:

- Choosing the right frameworks is a strategic decision, which can influence long-term business requirements, development, deployment and maintenance.
- The programming is tightly related to the frameworks. The implementation of B2B e-procurement in Chapter 6 has shown that Java codes are based on XML message formats. If the formats change, the Java codes have to change greatly (see 5.7.3).

Secondly, the factors that affect choosing the right frameworks include:

- Business partners’ selection. You should choose the same frameworks as your main business partners.
- ebXML and UDDI are at the top level and are important. Probably you can register with one UDDI Operator while following the development of ebXML.
- UDDI, led by Ariba, Microsoft and IBM, is tightly linked with SOAP and WSDL. If you choose UDDI, you might have to select SOAP and WSDL. Because BizTalk is the solution of Microsoft and cXML is the solution of Ariba, you also may have to consider BizTalk and cXML if you choose UDDI.
- Chooses the frameworks of XML messaging close to your requirements. In Figure 8, we mention four special types of frameworks, specific business operations, EAI,
vertical industry and special applications, you should follow the one close to your focus and follow its development. In the meantime, you have to pay attention to whether to adopt ebXML and/or UDDI.

- Selecting the SOAP is unavoidable because UDDI and ebXML have taken SOAP as the messaging approach. In the mean time, you need to pay attention to XML Protocol by W3C, which might replace SOAP.
- If the operating systems you and your customers use are Windows serials, BizTalk should be prioritized.
- Industrial vocabulary. If multiple initiatives exist, you need to compare them from the perspectives of feasibility, popularity, influential power of main players, and integration with upper layers (ebXML and UDDI).

Finally, it is noteworthy to pay special attention to some special frameworks:

- ebXML: to meet a deadline (May 2001); complexity of implementation; integrate current industrial standards; and industrial acceptance.
- UDDI: dependency on the technology of Ariba, IBM and Microsoft; visible business value (rate on investment).
- Vocabulary, syntax and semantics: pay attention to the XML repository (www.xml.org).

### 6.4.4 How to start

Although there are some problems associated with adopting XFIC, the benefits of XFIC are obvious for higher interoperability. If you prepare for their application, you can start with the following activities because they are not likely to change in the short term:

- SOAP. You can use SOAP for RPC calls between Web services, understand how they are structured and encoded for marshalling and unmarshalling, and explore the possibility of combining SOAP with your current Web server. Most commercial Web servers (e.g., Websphere and BEA WebLogic) support SOAP messaging. One SOAP implementation in Java is Apache SOAP (http://xml.apache.org/). If your Web server does not support SOAP, you can download Tomcat instead (see 5.5.1).
- XML parser using DOM and SAX (Simple APIs for XML). No matter what frameworks you select, you have to program XML processing to meet your business requirements. The method in selecting an appropriate XML parser is covered in 5.5.1.
- XML/CSS/XSLT. XML separates content and presentation style. You can use XML and CSS to create Web pages supported by Microsoft IE 5.0 and Netscape 4.7.3. You can try to transform XML into HTML or other texts using XML and XSLT. One XSLT implementation in Java is Apache Xalan (http://xml.apache.org/xalan/).
- UDDI interfaces. You can use the UDDI interfaces to register your Web services to present your business services to your customers and suppliers.
• WSDL. You can use WSDL to create your XML documents to represent your Web services for registration. Some commercial tools are available to facilitate the process.

6.5 Future research

We suggest two projects:

1. How can we combine XFIC with the existing middleware technologies (CORBA, EJB, RMI, MOM, etc) in B2B e-commerce development? XFIC can not exist independently, because they are used to integrate applications that may use one of middleware technologies. It is hard to abandon all past investments to capture XML technology. We need to find a way:
   • To correctly evaluate XML and XFIC and put them into a perspective to avoid an XML hype.
   • To efficiently combine XFIC with the existing technologies for e-commerce development.

2. Offer XML-based frameworks for auction-based pricing, i.e., how to model business collaboration, transaction and message structure and most importantly, offer some dynamic algorithms to determine suppliers. One benefit is that you can develop a Web service such as an ASP to serve for purchasing companies and B2B marketplaces. Another benefit is to allow applications conducting auction-based pricing to interoperate.
References


[34] Moreau, J. and Y. Fablet. Using XML for describing Web services, USA; April 11-12, 2001. Available at http://www.w3.org/2001/03/WSWS-popas-paper55


[42] Scannell, E. and Sullivan, T. *Oracle joins UDDI*, InfoWorld, 00-12-01. Available at http://www.infoworld.com/articles/hi/xml/00/12/01/001201honoracudii.xml


[48] Sullivan, T. *Web services: the ASP is being replaced by the Web services provider.* *How com?* 010312. Available at http://www.infoworld.com/articles/hn/xml/01/03/12/010312hnwebserv.xml


XML-based frameworks for Internet commerce and an implementation of B2B e-procurement

Yuxiao Zhao

It is not easy to apply XML in e-commerce development for achieving interoperability in heterogeneous environments. One of the reasons is a multitude of XML-based Frameworks for Internet Commerce (XFIC), or industrial standards. This thesis surveys 15 frameworks, i.e., ebXML, eCo Framework, UDDI, SOAP, BizTalk, cXML, ICE, Open Applications Group, RosettaNet, Wf-XML, OFX, VoiceXML, RDF, WSDL and xCBL.

This thesis provides three models to systematically understand how the 15 frameworks meet the requirements of e-commerce. A hierarchical model is presented to show the purpose and focus of various XFIC initiatives. A relationship model is given to show the cooperative and competitive relationships between XFIC. A chronological model is provided to look at the development of XFIC. In addition, the thesis offers guidelines for how to apply XFIC in an e-commerce development.

We have also implemented a B2B e-procurement system. That not only demonstrates the feasibility of open-source or freeware, but also validates the complementary roles of XML and Java: XML is for describing contents and Java is for automating XML documents (session handling). Auction-based dynamic pricing is also realized as a feature of interest. Moreover, the implementation shows the suitability of e-procurement for educational purposes in e-commerce development.

XML, e-commerce, framework, Web service, business-to-business, procurement, SOAP, open-source, integration
No 17  
Vojin Plavsic: Interleaved Processing of Non-Numerical Data Stored on a Cyclic Memory. (Available at: FOA, Box 1165, S-581 11 Linköping, Sweden. FOA Report B30062E)

No 28  

No 29  

No 48  

No 52  

No 60  

No 71  

No 72  

No 73  
Ola Strömors: A Structure Editor for Documents and Programs, 1986.

No 74  

No 104  

No 108  

No 111  

No 113  

No 118  

No 126  

No 127  

No 139  

No 140  

No 146  

No 150  

No 165  

No 166  

No 174  

No 177  
Peter Åberg: Design of a Multiple View Presentation and Interaction Manager, 1989.

No 181  
Henrik Eriksson: A Study in Domain-Oriented Tool Support for Knowledge Acquisition, 1989.

No 184  

No 187  

No 189  

No 196  

No 197  

No 203  

No 212  

No 230  

No 237  

No 250  

No 253  

No 260  

No 283  

No 298  

No 318  

No 319  

No 326  

No 328  

No 333  

No 335  

No 348  

No 352  

No 371  

No 378  

No 380  
Johan Ringström: Compiler Generation for Parallel Languages from Denotational Specifications, 1993.

No 381  

No 383  

No 386  

No 398  
No 488 Eva Toller: Contributions to Parallel Multiparadigm Languages: Combining Object-Oriented and Rule-Based Programming, 1995.
FHS 8/95 Dan Fristedt: Metoder i användning - mot förbättring av systemutveckling genom situationell metodkunskap och metodanalys, 1995.
No 538 Staffan Flodin: Efficient Management of Object-Oriented Queries with Late Binding, 1996.
No 546 Magnus Werner: Multidatabase Integration using Polymorphic Queries and Views, 1996.
Fif-a 1/96 Mikael Lind: Affärsprocessinriktad förändringsanalys - utveckling och tillämpning av synsätt och metod, 1996.
No 558 Patrik Nordling: The Simulation of Rolling Bearing Dynamics on Parallel Computers, 1996.
No 563 Niclas Andersson: Compilation of Mathematical Models to Parallel Code, 1996.
No 587 Jörgen Lindström: Cheferans användning av kommunikationsteknik, 1996.
No 589 Esa Falkenroth: Data Management in Control Applications - A Proposal Based on Active Database Systems, 1996.
No 591 Niclas Wahllöf: A Default Extension to Description Logics and its Applications, 1996.
FiF-a 4 Carita Åbom: Videomötesteknik i olika affärs situationer - möjligheter och hinder, 1997.
No 627 Fredrik Eklund: Declarative Error Diagnosis of GAPLog Programs, 1997.
No 653 Mats Gustafsson: Bringing Role-Based Access Control to Distributed Systems, 1997.
No 668 Per-Ove Zetterlund: Normering av svensk redovisning - En studie av tillkomsten av Redovisningsrådets rekommendation om koncernredovisning (RR01:91), 1998.
FiF-a 16 Marie-Therese Christiansson: Inter-organisatorisk verksamhetsutveckling - metoder som stöd vid utveckling av partnerskap och informationssystem, 1998.
No 742 Pawel Pietrzak: Static Incorrectness Diagnosis of CLP (FD), 1999.
No 766 Martin V. Howard: Designing dynamic visualizations of temporal data, 1999.
No 787 Charlotte Björkegren: Learning for the next project - Bearers and barriers in knowledge transfer within an organisation, 1999.
No 807 Svein Bergum: Managerial communication in telework, 2000.


Jean Paul Meynard: Control of industrial robots through high-level task programming, 2000.


Magnus Kald: The role of management control systems in strategic business units, 2000.


