Architecture for Data Exchange Among Partially Consistent Data Models

By

Eswar Venkat Ram Prasad Vemulapalli

Submitted to the Department of Civil and Environmental Engineering
in Partial Fulfillment of the Requirements for the Degree of

MASTER OF SCIENCE
at the
MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 2002

© Massachusetts Institute of Technology, 2002. All Rights Reserved.

Signature of Author ....
Department of Civil and Environmental Engineering
March 21st, 2002

Certified by ....
Steven Richard Lerman
Professor of Civil and Environmental Engineering
Class of 1922 Distinguished Professor
Director, Center for Educational Computing Initiatives

Accepted by ....
Oral Buyukozturk
Chairman, Department Committee on Graduate Studies

MASSACHUSETTS INSTITUTE OF TECHNOLOGY
JUN 3 2002
LIBRARIES

BARKER

Architecture for Data Exchange Among Partially Consistent Data Models

Eswar Venkat Ram Prasad Vemulapalli

Submitted to the
Department of Civil and Environmental Engineering

March 21st, 2002

in Partial Fulfillment of the Requirements for the Degree of
Master of Science

Abstract

With recent globalization, more and more organizations are having to exchange data through various means with the Internet playing a primary role. The Internet is increasingly being used as a global infrastructure for data exchange between autonomous participants. One of the biggest challenges facing organizations today is integrating the multitude of different information systems that have been implemented over the years. The problem with these kinds of inter-organizational data exchanges is that they involve a large number of information systems, which do not necessarily share a consistent data model. They require the ability to exchange semi-structured data.

Current practices to address this problem have been to get the participants involved in the data exchange to adapt a standard template for their autonomous data stores so that everyone understands each other. A more conventional approach was to get every organization to integrate their applications with each other, which is a very resource consuming exercise.

This thesis discusses the use of XML technologies for mapping information between partially consistent data models. The role of XML in semi-structured data exchange is described together with its application as a framework for data exchange. A description of an XSLT based architecture, which will take the unshared XML schema elements of these data models and map them, is outlined. A directory service that provides for the location of a suitable conversion resource such as XML-RPC / SOAP for satisfying the second stage of the discovery process is also described.

Thesis Supervisor: Steven R. Lerman
Title: Professor, Department of Civil and Environmental Engineering, MIT
Director, Center for Educational Computing Initiatives, MIT

Acknowledgements

First and foremost, I would like to thank my thesis supervisor, Professor Steve Lerman for his support and the confidence he put in me and in my work during my time in the Center for Education and Computing Initiatives (CECI) and MIT in general. During the duration of my thesis, he had been carrying out various roles at MIT such as occupying the faculty chair to heading CECI. He was also on a sabbatical for a semester. Despite such busy schedules, which come with his nature of work, he was always there when I needed him for advice and guidance. Prof. Lerman continued to be everything a graduate student could hope for in a thesis advisor and much more: supportive, understanding, challenging and flexible, all when appropriate. I cannot thank him enough.

Judson Harward, the principal scientist at CECI and my research supervisor during my work on ATIRP. He was tremendously helpful and supportive throughout the project and played a key role in the team producing a white paper and publishing a conference paper.

I would like to thank Kirky Delong, my research team partner who gave me plenty of guidance in my research work and without whose help I would not have been able to make a successful presentation of the paper at the ATIRP conference.

I would also like to thank all the staff and RAs in CECI for making the lab a very nice place to stay and study.

My two and a half years at MIT would not have been the same without the many number of close friends that I now call family. They are the closest to a family I have here in the US. Every one of them deserves mention in detail, but I would be remiss for not singling out a few.

- Anup Mantena, my roommate of two years whose sole acquaintance would have been enough to term my stay at MIT as truly satisfying and successful.
- Hiran Sammeta, my best friend outside of MIT who has given me a flavor of what life is beyond the shadows of the Charles River.
- Bharath Krishnan and Anand Ganti with whom I have shared many memorable experiences one of them being able to join me on a 6000-mile road trip to Texas and back on a day's notice.
- Debashish Sahoo and Balaji Rao with whom I could speak anything under the sky without any inhibitions and not regret it later.
- Neeta Kumari Singh, whose friendship I have had for the last three weeks, for making me feel acquitted with prolonging the masters degree by 6 months if only to have had the chance to meet her before leaving the east coast.

Last, but not the least, I would like to thank my family. My parents and sister continue to be everything to me and I could not have asked anything more from them.

Table of Contents

List of Figures

1.0 INTRODUCTION

The globalization of the architecture, engineering and construction industry has added new dimensions to the construction industry. Wide spectrums of technologies, particularly the Internet are being used by managers to manage these geographically distributed projects.

The Internet is one of the fastest growing, most exciting technologies in the 21st century, with many organizations participating in data exchanges with very large number of autonomous data stores. With an easy to use graphical Web browser, a project manager can access a wealth of information free or almost free (except for the fee for connection). One of the distinguishing characteristics of the Web is the fact that it is accessible at any time, from any place, to any number of users, with no third party involvement necessary. It is this unique quality that makes the Web such an ideal tool for the dissemination, gathering and analysis of information. The potential of Internet is enormous. Consider a scenario where a user or an organization only needs to update any information on the Web once to be accessible by all the people who are concerned with that information. For example, a project manager who is looking at the project schedule on the web browser will have the confidence that he or she is looking at the latest, up to the minute project schedule, which is impossible with traditional communication channels such as mail, fax, phone call or even email.

The major problem in realizing the potential of the internet with large and global projects is the involvement of a large number of information systems. While some of these information systems or autonomous data stores can be expected to share a consistent external data model, many will not and yet will still require the ability to exchange semi-structured data through some channel.

Therefore, an information-sharing and mapping model architecture for the integration of design and construction product and process information is necessary. There are usually two kinds of data exchanges: Intra-model and Inter-model. Intra-model data exchange occurs when similar groups within an organization transfer information between them. For example, a structural engineering consulting division might exchange information with the geotechnical consulting division of the same firm. In intra-model data exchange, the participants are assumed to share a data model and often an underlying software architecture. Therefore, the problem of data incompatibility in an intra-modal data exchange is relatively straightforward.

Inter-model exchange on the other hand takes place when unrelated groups within an organization communicate with each other and exchange data. Here, the participants will normally not share the same operational data model or software architecture as in the intra-model data exchange. An example is when a civil engineering construction company has a consulting division and a construction division, which need to communicate and exchange construction project information to various independent subcontractors. A typical construction project information may consist of drawings (e.g. AutoCAD™.), schedules (e.g. Primavera Project Planner™.) and databases (e.g. MS Access™ or MS SQL Server™.) A project planning tool will have to incorporate the product and process data, which may involve several of these information systems. The systems deployed for different aspects of the same project may be based on different algorithms and different data structures.

Inter-model exchange can also take place when two or more organizations communicate with each other and exchange data. This is the other problem that managers encounter while planning large-scale projects because these projects may involve as many as 300 different organizations. The complexity of these projects is likely to keep growing. The project information and the amount of data transaction among project participants tend to expand substantially. With the possibility of each one of these organizations having a different data model from the others, the integration of project information becomes a significant issue for the management of large-scale engineering projects. The communications across the various sub-disciplines of a large-scale construction project is inefficient and ineffective due to the inflexibility of the current data exchange. Inter-model data exchange often involves semi-structured, partially consistent data at best. Participants must be able to extract partial understanding from messages that depend on these inconsistent data models. The misinterpretation of documents and drawings can lead contractors to employ inappropriate construction methods, set up infeasible schedules, waste resources and misestimate project cost, etc.

The problem of inconsistent information can be illustrated by a case in where a simple confusion over weather measurements were metric or not led to the loss of a multi-million dollar spacecraft as it approached Mars in 1999. Preliminary investigations into the incident revealed that engineers at the Lockheed Martin Corporation, which had built the spacecraft, measured the thrust in pounds while the scientists in NASA thought the information was in the metric measurement of newtons. The assumed figure in newtons was incorporated into computer models that were used to calculate the spacecraft's position and direction. The resulting miscalculation let to the craft being off course by about 60 miles as it approached Mars. Although the data may have been wrongly provided, the real issue was that there was no process in place which could detect the discrepancy and correct it. If there were an automated process which could interface between the Lockheed Martin units of measurements and the NASA standards, then conversion of the values as described above would not have been an issue.

This problem of data compatibility between information systems is trivial if the organizations participating in the data exchange share a consistent data model. However in order to achieve a consistent shared data model, participants must agree on

1. the categories of data to be exchanged and their names,
2. the semantics of those categories, including controlled vocabulary and measurement units, as well as,
3. the syntax, protocols, and semantics for queries.

A consistent shared data model will lead to efficient data exchange and tightly coupled operational units, but cooperation cannot begin until the participants complete the entire architecture outlined above. Existing practices have seen consortiums formed for each industry. These consortiums outline guidelines for common standards, which each participant in the group needs to conform to so that they understand each other when data exchanges take place. However, this requires each participant to adapt their whole technology infrastructure to these standards, and smaller sized firms with limited financial resources are usually the first to fail in confirming to them.

While some of the participants in construction projects can be expected to share a consistent external data model, many will not and yet will still require the ability to exchange semi-structured data. This alternative approach will implement partial data exchange using semi-structured, usually tagged, data. The approach has the advantage that the architecture will put in place a process which will take this partially inconsistent data and convert it into a standard format which other participants in the data exchange process will be able to interpret and convert to a format they can understand. It also has the added advantage of putting in place an incremental data exchange process tailored to changing requirements and by feedback from the previous stage of the exchange.

The commercially off-the-shelf (COTS) Web standards including XML (Extensible Markup Language) and related technologies provide an excellent medium for exchanging semi-structured data and for brokering information exchange between organizations possessing autonomous data stores. These XML and XML-related standards provide several advantages in dealing with partially consistent data models in a non-intrusive manner, i.e. without requiring the different units to change their data implementations. XML is platform and technology independent. It also provides the flexibility required to express data objects from general data models. At the same time, it requires syntactic correctness, which in turn is necessary for verifying the correctness of the conversion schemes between the data models. On another level, XML-Schema's ability to define data types and structures of XML elements allows the mapping of these properties and relations from the inconsistent data models to the common realm of XML.

This thesis investigates strategies that exploit both common base schemas mapping of data types, element/attribute names and directory-based location of resources for data conversions. The goal is to devise an architecture for exchange between distributed data stores that will support any number of participants irrespective of their diverse data models. The architecture will suggest a way that will assist organizations with limited financial resource to take part in data exchanges with other larger firms without having to conform to their standards.

In the following chapter, this paper provides some background into some of the technologies used in the proposed architecture as well as discusses some existing commercial practices. The literature review suggests that frameworks, based on principles which minimize the risk of updating a whole organization's technology infrastructure due to continuously evolving standards and technology have not been thoroughly looked into. This thesis makes a rudimentary and yet a significant step in proposing a new approach. We call this architecture InfoX architecture. To explain this architecture, we need to explain two key aspects of the technology used. They are network protocols and directory services and are explained in Chapter 3 and 4 respectively.

In chapter 5, we discuss the InfoX architecture in details. We also discuss the benefits of this architecture over other existing commercial models. To ensure that the framework is not only good in theory, but also implementable, we have addressed issues of maintainability, scalability, performance and security. The architecture can be extended in many ways, which are not explored because of limitations of time and expectations from a master's thesis. Therefore, in chapter 6, we conclude and describe the directions for future research and development in this area.

2.0 LITERATURE REVIEW

2.1 Autonomous Data Stores

The recent meeting of database researchers at Asilomar [Bernstein et al. 1998] emphasized the importance of the WWW as a federation of a potentially unbounded number of data stores, with many of these being embedded in "gizmos", that is, autonomous embedded systems in consumer electronics and the like. Information exchange in the 21st century will resemble the evolving web with very large numbers of autonomous data stores, often lacking any human control. While some of these can be expected to share a consistent external data model, many will not and yet will still require the ability to exchange semi-structured data through data discovery and negotiation. We feel this problem is isomorphic to data discovery and exchange on the emerging Semantic Web [Berners-Lee, 1998.3], an extension of the well-known WWW intended for machine-to-machine data exchange without a human mediator.

This emerging "business-to-business" architecture evolves from natural organizational behavior. Organizations instinctively protect their information to maintain security and autonomy. They do not want potential foes/competitors to anticipate their actions, and at the same time, they want the freedom to change their own organization and its corresponding data model without elaborate consultation with peer organizations. If General Motors is reorganizing a division it does not want to wait for Ford to sign off on the changes, just as the US Army resists external review, even by allies, of the details of organization, deployment, and readiness. This tendency towards organizational independence is balanced by the demands of external cooperation. Any such cooperation with peer organizations requires information exchange. In our information rich world, efficient information exchange almost always requires the computer-mediated trading of semi-structured information.

Overcoming the natural desires for security and autonomy that divides the divisions of the same company require immense effort. The COTS Web standards including XML (Extensible Markup Language) and related technologies provide an excellent medium for exchanging semi-structured and for brokering information exchange between organizations possessing autonomous data stores.

2.2 Functional Views of XML Technologies

2.2.1 XML as Document Markup for the WWW

The original WWW document model as specified by HTML (Hypertext Markup Language) paid homage to the tradition of tagged markup languages that emerged during the 1970's and 1980's. As WYSIWYG document editors proliferated with the rise of the personal computer, a reaction set in that led Brian Kernighan to declare, "The problem with 'What you see is what you get' is that what you see is *all* you get." By focusing simply on the appearance of the printed page, these document editors had dropped any sense of the document's organization. This realization led to systems like TeX and LaTeX that attempted to separate the appearance of the document from its logical structure. The goal was to break the document into semantic units like "ChapterTitle" or "BibliographicalEntry", and then specify how each particular semantic unit should appear on the printed page.

One problem with this approach is that each field and document type requires its own set of semantic tags. A dictionary requires different tags from a sales catalog or a technical manual. The climax of this trend is a meta-language called SGML (Standard Generalized Markup Language) that was designed to specify field-specific sets of hierarchical document tags. Each such set of tags formed a *document type definition* or DTD. True SGML is little used outside of the publishing industry, but HTML originated as an SGML tag set for web documents. The rapid evolution of HTML to meet user requirements and the competition between browser and other tool vendors doomed the purity of this approach. HTML focused on how a document appeared in a browser rather than on delineating the internal structure of the document. It evolved to please the web surfer's eye rather than systematically tagging data for machine-to-machine exchange.

SGML's large and complex feature set has also hindered its limited acceptance. Its complexity makes SGML a versatile environment, but it also complicates the task of those who develop SGML implementations and SGML-based toolsets. XML arose out of an effort by the W3C Consortium:

1. to devise a clear separation of content organization and visual presentation for WWW documents, and
2. to design a simpler version of SGML for the WWW.

HTML in the mean time became a major markup language used widely across the world. Initially HTML started under-defined with proprietary extensions and incompatibility abounding at later stages. Attempts were then made to rein in HTML by providing a DTD, it turned out that several DTDs were needed to manage the variants. Hence, the XML namespace [1] mechanism was developed in part to allow more control of proprietary and standard extensions. The current strategy of the W3C Consortium and vendors is to rewrite the current version of HTML (4.0.1) as an XML DTD. This new, more rigorous HTML is called XHTML.

XHTML is the reformulation of HTML 4 as an application of XML. The hope is that it will both extend the life of HTML by putting it on a more extensible and platform-independent base as well as forming a bridge to the next generation of WWW documents based on a wide variety of XML DTDs. XHTML 1.0 is the basis for a family of document types that subset and extend HTML.

People recognized was that there was a missing layer required which would enable mix-and-match selection of components even within a namespace. From this realization came the XHTML Modularization project at the W3C. XHTML Modularization makes it convenient to create specialized versions of XHTML: subsets with tailored content models and extensions in other namespaces. The purpose of modularization is to allow someone, perhaps not an expert in DTDs or Schemas, to restrict and extend their own version of HTML. Using modules means they will not leave something out by accident, as well as that there are placeholders for extensions and restrictions that are convenient and visible to others. Therefore, modularization does not actually alter the expressive power of DTDs or W3C XML Schema. Instead, it provides an abstract model and practical conventions for how to organize a DTD or Schema.

As the abstract to the Recommendation Modularization of XHTML puts it,

'This Recommendation specifies an abstract modularization of XHTML and an implementation of the abstraction using XML Document Type Definitions (DTDs). This modularization provide a means for subsetting and extending XHTML, a feature needed for extending XHTML's reach onto emerging platforms.'

XHTML Modularization may become one of the most important new technologies of 2001.

2.2.2 XML for Semi-Structured Data Exchange

The previous section concentrated on the role of XML in providing support for human-readable documents on the WWW. But XML will probably exert greater influence as the enabling technology for a quantum leap in automated information exchange between networked computers on the WWW. This is currently the focus of great commercial interest. A whole new class of business-to-business XML-based applications has arisen to expedite inter-company information exchange without human intervention, thus establishing the model for the Semantic Web.

XML provides a hierarchical tagging structure that can be used to communicate data from multiple data models. It is well adapted for the robust transfer of data between relational databases, but its tree-based hierarchical structure also makes it appropriate for the communication of object-oriented data. Consider the following brief example of an XML description of a concrete mixing truck including its position and fuel remaining:

```
<CM-Truck id=4591 >
        <position>
                <lat>39.30.42</lat>
                <lon>-76.9.42</lon>
        </position>
        <fuel>238.7</fuel>
</CM-Truck>
```

The use of a DTD for data exchange allows the receiving XML parser to validate the information as to form, but the content may still be corrupt or nonsensical. That is, it might well fail standard database integrity constraints when the data is parsed. Therefore, the translation of XML formatted data to and from a host's internal data model is non-trivial. In an effort to simplify this task, the W3C and vendors have together developed standard APIs to govern the parsing of XML data. The simplest and earliest standard API is known as the SAX (Simple API for XML) API. SAX compliant parsers call a standard event driven API as an XML document is parsed. There are separate callbacks for the recognition of various XML syntactic units. Other parsers attempt to process an entire XML document producing an in-memory tree of nodes representing the various syntactic units of the document and their relationship to each other. The Document Object Model (DOM), a W3C standard, describes a second standard API for accessing this in memory tree and editing it. It is important to note, however, that the DOM standard does not cover the details of parsing or writing a DOM tree back out into an XML document stream.

More recently, Sun Microsystems has announced (but not released) a special XML parser code-named Adelard for the exchange of information from Java to XML and vice versa. In Adelard, the object-oriented data model is specified in an extension of XML called XML Schema (see below). The Adelard compiler then generates code to parse XML data in the data model, to validate it (to the degree that the validation criteria can be expressed in XML Schema), and then to create instances of Java objects to represent the parsed XML entities. The corresponding Java classes contain methods to marshal their instances into appropriate XML code. Since XML Schema allows the specification (and validation) of object type as well as range checking and other simple integrity checks, Adelard's automated "data binding" will facilitate the validation of data in machine-to-machine exchanges [2]. Of course, a programmer can extend the validation in the Adelard-generated code via arbitrary hand-coded methods. This Adelard-based approach called data binding should offer significant advantages in the application of XML to semi-structured data exchange.

2.2.3 XML Schema as a Framework for Data Discovery

As mentioned above, XML Schema is an extension of the XML standard that allows a variant of the DTD called a *schema* to define object-oriented data types using inheritance and certain validation criteria. The inheritance mechanism of XML Schema allows an organization to adopt standard schema definitions and adapt them for the particular data model(s) they use. If they then publish these schemas, the inheritance relationships can be used to recover part of the semantics of their data model. This approach possesses serious limitations in that the equivalence of fields (elements and attributes in XML) and data types ultimately depends on a matching or mapping of tag names.

2.2.4 XSL for Data Transformation

The Extensible Stylesheet Language (XSL) is a language for expressing style sheets. An XSL style sheet is a file that describes how to display an XML document of a given type. It includes both a transformation language, Extensible Stylesheet Language Transformation (XSLT) and a formatting language, each of these being an XML application. The transformation language provides elements that define rules for how one XML document is transformed into another XML document. The transformed XML document may use the markup and DTD of the original document or it may use a completely different set of elements. In particular, it may use the elements defined by the second part of XSL, the formatting objects.

Its ability to move data from one XML representation to another makes XSL an important component of XML-based electronic commerce, electronic data interchange, metadata exchange, and any application that needs to convert between different XML representations of the same data. These uses are also united by their lack of concern with rendering data on a display for humans to read. They are purely about moving data from one computer system or program to another.

There are three primary ways to transform XML documents into other formats with an XSLT style sheet:

- The XML document and associated style sheet are both served to the client, which then transforms the document as specified by the style sheet and presents it to the user.
- The server applies an XSLT style sheet to an XML document to transform it to some other format and sends the transformed document to the client.
- A third program transforms the original XML document into some other format before the document is placed on the server. Both server and client only deal with the transformed document.

Each of these three approaches uses different software, although they all use the same XML documents and XSLT style sheets. An ordinary Web server sending XML documents to Internet Explorer™ is an example of the first approach. A servlet-compatible Web server using the IBM alphaWorks' XML Enabler[1] is an example of the second approach. A human using Michael Kay's command line SAXON program[2] to transform XML documents to HTML documents, then placing the HTML documents on

[1] http://www.alphaworks.ibm.com/tech/xmlenabler

[2] http://users.iclway.co.uk/mhkay/saxon/

a Web server is an example of the third approach. However, these all use the same XSLT language.

While converting information from an XML document to another format, data contained in these XML documents can be processed in various ways to obtain the required form.

For example, functions calls could be made within the style sheet to make conversions to data. For example, temperature data may need to be expressed as Fahrenheit rather than degree Celsius.

2.3 Existing Commercial Practice

2.3.1 Biz Talk

One of the Microsoft products, Biz Talk Server, is a set of system software and development tools that use XML to solve two of the most intransigent problems corporations and governments face today: integrating internal applications by tying together their data streams and process logic, and integrating applications with supply chain partners to support ambitious e-business efforts (B2B). At its heart, the BizTalk server is a document hub. Its features include of data interchange, security, remote location data polling, document type mapping, rules-based business document routing, document interchange management, and document tracking and analysis. It employs XML as its internal data format.

All inbound documents are parsed and stored as XML, regardless of their format (EDI, delimited text, and so forth). Outbound documents are serialized from XML into the format appropriate for the receiver. BizTalk knows how to parse inbound and outbound data by following schemas that are shared by data-trading partners. BizTalk.Org, a consortium of user organizations and vendors, keeps available a range of schemas for various industries applications.

BizTalk server completes this process using a pair of function sets: Orchestration and Messaging. Orchestration handles all the business functions. It lets you create processes graphically and connect them to code capable of carrying them out. Messaging is a set of facilities that performs basic data integration functions such as data description and field mapping from one application to another. To make this process work, one needs to tell the system the data definitions of files you plan to use (fields, datatypes tec.); how to map fields from one data set to the other; and how to process data flows using which communications channels to which destinations and which, if any, imposed conditions.

2.3.2 Application Integration Services

One of the more conventional methods is System Integration of two separate organizations. Many system integration firms use their own process to achieve this goal. This is usually called Enterprise Application Integration (EAI). The main purpose of EAI is to replace independently maintained interfaces with a disciplined integration approach that is supported by EAI technology. Because of integration architecture, systems may be incrementally added to the infrastructure without invalidating other connections to the collaborating systems. This can allow for the growth of the integration system. The EAI process involves consulting firms to come in, evaluate and work on this process. This is usually very labor intensive, and hence can be quite expensive for the organization. In addition, for future customization work, they do have to depend on labor-intensive processes because of the lack of standards in this process.

There are also numerous tools provided by companies such as Web Methods, TIBCO, IBM, iPlanet that can accelerate this process of Business Integration. These tools form the middle layer when organizations try to exchange processes and data.

Even to use these tools, companies have to depend on the services of the business integrators to attain their goal. Since all the business integrators have proprietary technology, the customers will always have to depend on them. Also, these middleware tools force organizations to follow standards that are set by the system integration companies.

2.4 Proposed Framework Overview

This thesis investigates a framework that exploits both common base schemas in an inheritance hierarchy and directory-based location of webservices, which will provide information on data conversions. The goal is to devise an architecture for information exchange between distributed data stores that will support less resourced organizations to successfully take part in a collaboration. A pattern for data sharing that evolves through the following stages is proposed:

1. Two organizations recognize their need to exchange information. Both start a process to identify common or mapable elements in their public data models. These elements are then mapped to semantically equivalent elements in a common data model. An example of semantic equivalence is when for instance, an organization may use a field named *position* to designate what

another would call *location*. Directory services could be used to do provide for this mapping.

2. Both organizations then commence distributed queries to net-based directories to establish webservices, which will provide conversions for the fields in their public data models to a common data model format. This is called data equivalence. For example, organizations may record the same concepts using different but mutually convertible data types. This may be as simple as the confusion between newtons and foot-pounds that doomed a recent NASA Mars mission or may involve a more complex data conversion, say from latitude and longitude to Universal Transverse Mercator coordinates (UTMs).

Though the mapping search establishes a semantic equivalence, in order to use it, however, the two elements must employ identical data types or we must also find a conversion path from one data type to the other. This leads to establishing data equivalence.

However before any attempt is made to describe the proposed framework for any data exchange in detail, it is important to look at the network protocols and directory services which play key roles in the architecture. These are therefore described in the next two chapters.

3.0 NETWORK PROTOCOLS

When an organization wants to engage in an interaction of some kind with another individual or organization, there are two things that need to get sorted out up-front:

1) What is the structure and syntax of the language we are going to speak (i.e., what messages and data are we going to exchange, and how?);
2) What are the underlying semantics behind this language (i.e., in a real-world sense, what does it mean to give a value of 58 with a label of "GE" attached to it?)

This need for structure and semantics arises whether the interaction is between two people, a person and a computing device, or two computing devices. The context that has the most interest for researchers is the interaction between two computing devices, where money is involved in the conversation.

Until very recently, online interactions were typically done in one of two ways. A digital exchange can occur in a direct, tightly coupled connection, where the structure and syntax of the messages are encoded into object interfaces and the parties engage in remote method calls (over CORBA/IIOP, RMI, DCOM, etc.). Alternatively, the connection can be more loosely coupled, defined in terms of GET/PUT arguments on well-defined URLs (e.g., validate a credit card transaction by making a request in the form of POST arguments to a particular SSL-enabled URL).

Data exchanges are handled within these contexts in various ways: as method arguments; as URL arguments; as structured data streams generated from either of these sources; or sometimes even as out-of-band direct database transactions. The semantics of these exchanges can be local and customized, or in rare cases, there may be well-known, high-level APIs in play, such as an e-commerce component library or widely published and well-documented EDI (Electronic Data Interchange)-based protocols.

XML, arose to address the need for a common, flexible context for defining the structure and syntax of messages and data. This was really a return to the SGML roots of HTML, which by 1996 had many presentation-specific details incorporated into its syntax. In an XML context, Document Type Definition (DTDs) and XML schemas provide a well-defined format for specifying (and, more importantly, sharing) the structure and syntax of an exchange. The semantics and rules of the exchange are agreed upon as part of the ancillary elements of the DTD/schema documentation. For example, "a 'Position' tag will contain data representing the location of a concrete mix truck within five minutes of the time the tag was generated at the source," or, conversely, "when asked for a location, a compliant AcmeXML participant will respond with a well-formed 'Position' tag."

It is only natural to think of using XML in an online-messaging or remote-method context, and that is what happened next. XML-RPC (Remote Procedure Call) came around 1998 as a way to encode remote method calls and responses in an XML-based format, and as a way to transmit these remote method payloads over HTTP. Simple Object Access Protocol (SOAP) evolved out of the same work that created XML-RPC.

The interesting thing about this effort is that it is really a move back to the days before distributed object protocols were developed. The RPC protocol is a scheme for encoding remote procedure calls into a standard representation, then serializing these calls onto the wire and transmitting them to a remote RPC peer, where they are deserialized, processed, and results are similarly encoded and returned. Distributed object protocols came about as a way to dissolve the interface between RPC capabilities and object-oriented environments like Java and C++. Once the up-front work is done to define a remote object and implement its methods, remote method calls are made in the code by calling methods on remote object "stubs," which are obtained from a remote service. No more complications with RPC encodings of method arguments and responses: The distributed object system handles all this when a method call is made on a remote object stub.

SOAP simply uses XML as an encoding scheme for sending request and response parameters with the help of HTTP as a transport. It consists of a small number of abstractions like the SOAP method, which simply is an HTTP request, and response complying with the SOAP encoding rules. XML-RPC and SOAP roll the clock back to RPC, then move it forward again using XML as the encoding context instead of object interfaces. Then they specify a way to deliver XML-encoded data and RPCs over HTTP. The idea here is to encapsulate the services at a different level and export an XML face to the world, rather than object interfaces.

Stepping back for a moment, it is worth asking why we need a protocol like SOAP at all. Given that a web service involves exchanging information encoded in XML, there is nothing to stop two parties from agreeing on a given XML vocabulary and structure, effectively defining their own protocol. However, this means that each pair of endpoints essentially defines an ad hoc protocol. Therefore, given *n* endpoints the potential number of protocols is $n(n-1)/2$. While implementing any single protocol may be a reasonably simple task, when *n* is of a significant size the implementation burden becomes quite significant. Having a standard protocol, rather than many ad hoc protocols, eases the implementation burden by bringing uniformity to certain aspects of communication. This ease of implementation leads, in turn, to processing facilities being built into other software, for example, server products, client products, toolkits, and operating systems. This frees the implementer of a web service (or clients of the service) to concentrate on the pieces specific to that service, rather than on the generic pieces that all web services require.

XML Web Services are being hailed by the industry as the enabler for freeing information from the confines of HTML. Using SOAP, data can be encoded in XML and transmitted using any number of Internet protocols. So long as both the sender and the receiver can agree upon the message format—that is, the protocol that SOAP defines—information can easily be exchanged in a platform-independent manner. An organization Web service can receive a SOAP payload from a remote service, and the platform details of the source are entirely irrelevant. Anything can generate XML, from Perl scripts to C++ code to J2EE application servers. So, as of the 1.1 version of the SOAP specification, anyone and anything can participate in a SOAP conversation, with a relatively low barrier to entry.

The following request is an example of a SOAP message embedded in a HTTP request. The complete code is shown in Appendix A.

Organization A is requesting the namespace identification of Organization B.

```
Host: 209.110.197.12 // address of computer from where the request is  made.
SOAPMethodName: "URL"#getidentification
// declaring the name of function which will be used in the SOAP message body.
 <se:Body>
 <m:getidentification xmlns:m="URL">
        <org>OrgB</org>     // requests the namespace identification of Org. B
  </m:getidentification>
</se:Body>
```

Following is the response message from OrgB, containing the HTTP message with the SOAP message as the payload.

```
HTTP/1.1 200 OK     // response is  successful
 <se:Body>
   <m:getidentificationResponse xmlns:m="URL">
    <result>url//xxxx</result>     // the namespace of Organization B is returned
  </m:getidentificationResponse>
 </se:Body>
```

The following SOAP message is sent to Org B requesting for the details of the object Destroyer

```
SOAPMethodName: "URL"#getobjectdetails  // function name that is being used
  <se:Body>
    <m:getobjectdetails xmlns:m="URL">
            <objID>5678</objID>          // requests details of object# 5678
    </m:getobjectdetails>
  </se:Body>
```

Following is the response message from OrgB, containing the HTTP message with the SOAP message giving the name and schema identification of the object destroyer.

```
<se:Body>
  <m:getobjectdetailsResponse xmlns:m="URL">
    <objname>url//xxxx</objname> // returns the object name
    <schema>yyyyy</schema>       // returns the schema name it belongs to
  </m:getobjectdetailsResponse>
</se:Body>
```

Some SOAP servers will map RequestURIs to class names, dispatching the call to either static methods or to instances of the class that live for the duration of a request. Other SOAP servers will map Request-URIs to objects that are kept live over time, often using the query string to encode a key.

3.1 Pros and Cons of using SOAP

One of the major aspects that has led to SOAP gaining popularity is its simplicity in accomplishing remote object/component/service communications (hence its name). It formalizes the vocabulary definition in a form that is now familiar, popular, and accessible (XML). If one knows XML, it is easy to figure out the basics of SOAP encoding quickly. In these regards, SOAP has an edge on the predominant remote object protocols (RMI, IIOP, DCOM). RMI is very straightforward if one knows Java, but it requires Java running on both ends of the connection. Hence, it puts additional platform restrictions on the participants. CORBA decouples the protocol from the runtime environment but its framework is relatively complex, and there is a learning curve to invest in before adapting enterprise-wide CORBA systems. Microsoft COM/DCOM also has platform restrictions.

However, the real dividing line is how the vocabulary is defined between the parties. In the case of RMI, CORBA, and DCOM, how to speak to a remote service is encoded in the object interfaces that it exports, along with the semantics defined behind these interfaces. So one has to know and understand the Java, IDL, or MIDL definitions for these interfaces in order to interact with them. With SOAP, one still needs to know the interface to your service (What requests do you respond to? What data types can you understand and recognize?), but the interface can be given to the others in the form of XML.

As the saying goes, nothing is perfect. One should be aware that SOAP has its fair share of imperfections. The SOAP specification contains no mention of security facilities. One of the advantages of SOAP is that it runs over HTTP, which eliminates firewall problems. No enterprise will want to open up a channel to make direct, unprotected method calls on their Web services. Some will build custom security measures on top of SOAP, to ensure that authentication, authorization, and accountability are preserved. This tosses a great deal of the interoperability of SOAP out the window.

Others will fill this security gap at the network level, sniffing HTTP traffic passing through their firewalls, and restricting SOAP payloads to privileged IP addresses and ports. However, this leads to a trade off of portability due to the network administration overhead. If and when SOAP payload filters become common services from firewall vendors, this overhead will go away. But this only makes sense if SOAP traffic is well defined and detectable.

However, this leads to another issue. The current version of the SOAP specification (1.1) does not specify a default encoding for the message body. There is an encoding defined in the specification, but it is not required that one use this encoding to be compliant; any custom encoding that is chosen can be specified in the encodingStyle attribute of the message or of individual elements in the message. The default encoding spelled out in the spec may become a de facto standard by SOAP implementations, but the standard needs to be made it explicit so that SOAP interoperability can be well-defined and testable.

As vendor activity heats up around this, there is every possibility that vendors will start to use the "SOAP compliant" label rather loosely. If they do indeed start to use custom encoding styles, the adoption of SOAP will suffer from lack of interoperability. This may sound like the mistakes made with CORBA in its early days by not specifying a standard wire protocol. CORBA suffered for this lack of interoperability, and SOAP may run the same risk by leaving this hole unfilled.

SOAP is simple, accessible, and very portable but there are various trade-offs involved in its use. SOAP is very simple compared to RMI, CORBA, and DCOM because it does not deal with certain ancillary but important aspects of remote object systems. There is a lack of security provisions. In addition, the specification itself explicitly excludes distributed garbage collection, objects-by-reference, and remote activation as being not part of the core SOAP specification. The SOAP model also does not include any provisions for object lifecycles, session/state management, or distributed transactions. There are ways to add custom header entries to a SOAP message to address some of these issues (the SOAP specification includes examples that show custom transaction-oriented header fields), but these custom services layered on top of the standard SOAP model also severely limit interoperability.

If SOAP is expanded to include all of these services, it would bloat significantly, and would get much more complicated. Rather, it needs to expand to include the strictly necessary elements, like security and perhaps distributed transactions. Moreover, for the other issues, like component lifecycles and session management, the SOAP specification could be amended to include the underlying assumptions about the models in use by SOAP agents. In other words, define the high-level contracts that SOAP participants need to satisfy with their underlying implementations and leave vendors and developers the freedom to continue to use the tools that they want, as long as they honor the specified contracts. SOAP needs to stay as simple as possible, and it does this by limiting its target domain to messaging situations where its simplified runtime model is sufficient. The upshot of all this is that SOAP offers industry support for a new suite of design and implementation patterns, and a way to quickly establish interactions between online services.

4.0 DIRECTORY SERVICES

The most familiar kinds of directories are the ones we use in our everyday lives such as the yellow pages or TV guide. These are called offline directories. The directories in the computer and networking world are similar in many ways but with some important differences. These directories are called online directories and are different in the following ways. Online directories are

Dynamic: They are up-to-date with information and are timely maintained by administrators. Sometimes, administrative procedures are put in place to update the directory automatically so that whenever there is a change, it is reflected immediately to users.

Flexible: They can easily be extended with new types of information with minimum additional cost. They are typically designed to be extended without a need for a redesign. Another way they are flexible is by supporting several kinds of data organization simultaneously therefore providing more advanced types of searches.

Secure: Directories centralize information, allowing access to that information to be controlled. Clients accessing the directory can be identified through a process called authentication. The directory can use the identity established in conjunction with access control lists (ACLs) and other information to make decisions about which clients have access to what information in the directory.

Personalized: By identifying users who access the directory and profiling information about them, directory services can easily provide personalized views of the directory to different users. The personalization could be based on interests explicitly declared or could be based on the client's previous interactions with the service.

A directory service provides a way to manage the storage and distribution of shared information. Directory services are simple databases and hence provide search and filter functionality. Instead of locating an entry only by name, these directory services allow locating entries based on a set of search criteria. Naming services and directory service provide name to object mapping, and directory services provide information objects and tools for searching for them.

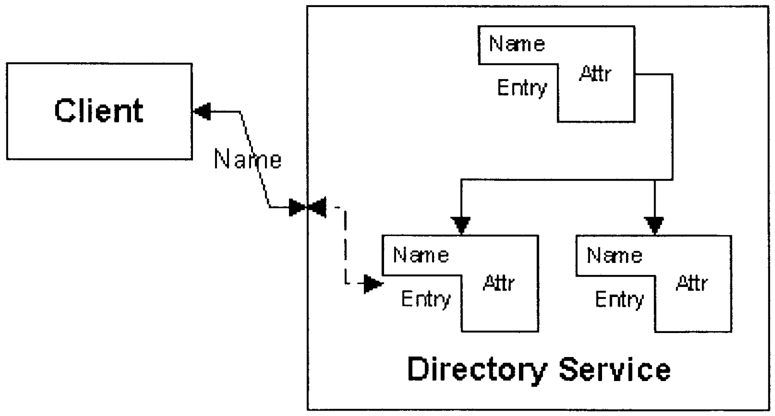


Figure 4-1: **Naming & Directory Service**

Directory services, long overlooked, are becoming critical components of an organization's overall information systems infrastructure. As information systems and networks continue to evolve and grow, applications and users are becoming more dependent upon access to some type of directory information. Activities associated with the movement of people throughout an organization affect many different systems and databases, each with its own directory and administration interface, resulting in inconsistency in information.

Early network directories were most often developed specifically for a particular application. In these proprietary directories, system developers had little or no incentive to work with any other system. But systems users, in an effort to rationalize their ever-increasing workload, sought ways to share access to and maintenance of directory databases with more than one application. This dilemma engendered the concept of the directory as a collection of open systems that cooperate to hold a logical database of information. In this view, users of the directory, including people and computer programs, would be able to read or modify the information or parts of it, as long as they had the authorization to do so.

This idea grew into the definition of X.500 [3]. Although the X.500 standard coverage was comprehensive, implementers have criticized it as being too complex and therefore too difficult to implement. Lightweight Directory Access Protocol (LDAP) offers much of the same basic functionality as X.500 and can be used to query data from proprietary directories as well as from an open X.500 service. Although LDAP started as a simplified component of the X.500 Directory, it is evolving into a complete directory service. It has matured and has added features not found in X.500 and moved into areas not addressed by the older spec, like APIs and data formats.

Unfortunately, most organizations today are in a state of directory chaos, with multiple islands of single purpose directories all separately maintained. As directories continued to expand within an organization, additional problems arose. Enterprises often found themselves with multiple occurrences of each type of directory, with no easy, cost-effective way to achieve directory integration. In addition, the movement of people between locations and departments required the need to access, change and maintain the affected directories, often increasing the overall workload across an organization and resulting in duplicated effort.

Finally, since directory services were typically implemented on an application-by application basis, there was no single organizational entity responsible for maintaining an enterprise's directory services. Instead, directory services were splintered among multiple support groups, causing not only technological integration issues, but also inconsistent directory information, and the political issues associated with who owns the enterprise's directory services.

Organizations are beginning to tackle the problem of integrating these disparate directory services into an enterprise-wide service. For many organizations, the current best-case scenario is to consolidate all of their disparate directories into one of each type of directory. This is a first step towards the ultimate goal- a single, all-purpose directory service that supports all systems, applications, and devices across the enterprise.

Recent advances in directory services technology have enabled organizations to begin devising an overall direction for creating an integrated, enterprise-wide directory service. The widespread adoption of the Lightweight Directory Access Protocol (LDAP) by vendors is providing a cornerstone for this integration. An integrated directory service provides the opportunity to reduce the number of directories to manage and maintain, minimize the data entry points for duplicate information and provide a single point for the administration of configuration information with a device or user.

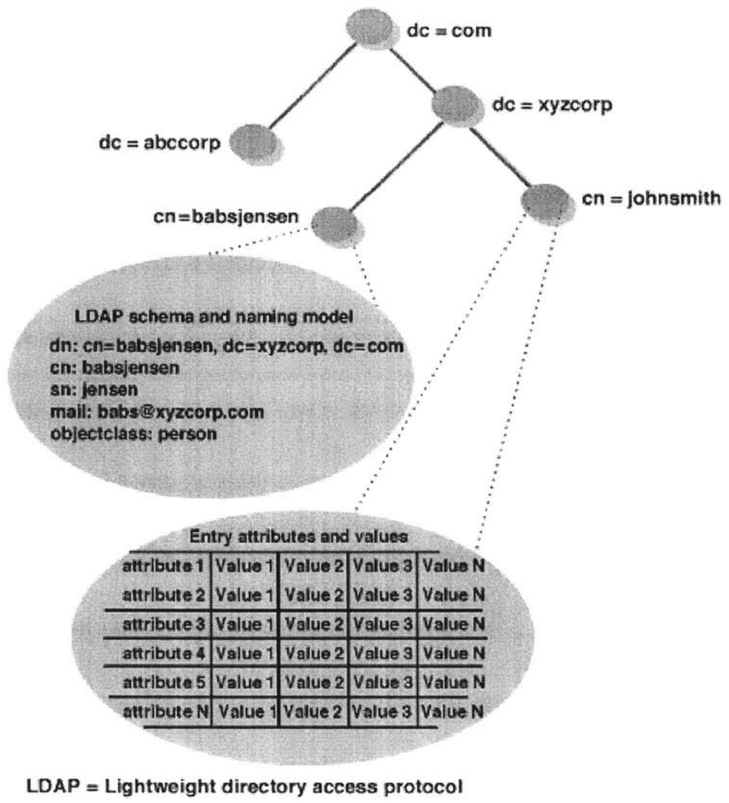
Using LDAP, an enterprise can develop a single, logical directory service. This does not necessarily imply a single, physical directory server. Instead, the directory service will most likely be comprised of physically distributed directory servers that each supports a specific domain. However, the difference between this and the chaos that currently exists is that distributed directory servers will be able to query one another for information about users and devices using LDAP. The net result is a collection of directories that function like a single, integrated directory service that can be administrated easily and centrally.

The current specification of LDAP comprises of features and functions for defining directory-related tasks like storage and retrieval. The information model is inherited almost unchanged from X.500 directories and is organized according to collections of attributes and values known as entries. The model is extensible with the ability to add any kind of new information to a directory. LDAP schemas define the actual data elements that can be stored in a particular server and how they relate to real world objects. The collections of values and attributes representing objects such as organizations, departments and groups are defined in the standard, and individual servers can also define new schema elements. The LDAP naming model is hierarchical with the individual names being composed of attributes and values from corresponding entries, while the LDAP functioning model determines how clients access and update information in an LDAP directory and how the data can be manipulated. It offers some basic functional operations such as add, delete, modify, search, compare and modify DN (distinguished name). Add, delete, and modify operations govern changes to directory entries, while search locates specific users or services in a directory tree. The compare operation allows client applications to test the accuracy of specific information using entries in the LDAP directory, while the modify DN operation makes it possible to change the name of an entry.

LDAP protocol specifies the interaction between clients and servers and determines how LDAP requests and responses are formed. The application program interface (API) details how the client applications access the directory, providing a standard set of function calls and definitions.

4.1 Directory Structure

An LDAP directory is structured as simple tree hierarchy, which conforms to the LDAP schema and naming models. The naming model is needed to give a unique name for any entry into the directory, allowing reference to any entry unambiguously. In LDAP, distinguished names (DNs) are used to refer to entries.



LDAP schema and naming model
dn: cn=babsjensen, dc=xyzcorp, dc=com
cn: babsjensen
sn: jensen
mail: babs@xyzcorp.com
objectclass: person

Entry attributes and values

attribute 1	Value 1	Value 2	Value 3	Value N
attribute 2	Value 1	Value 2	Value 3	Value N
attribute 3	Value 1	Value 2	Value 3	Value N
attribute 4	Value 1	Value 2	Value 3	Value N
attribute 5	Value 1	Value 2	Value 3	Value N
attribute N	Value 1	Value 2	Value 3	Value N

Figure 4-2: LDAP Directory Structure

The topmost (root) node is typically the domain name component (dc) for a company, state, or organization. Below that are entries for organizational units, like branch offices and departments, followed by common name (cn) entries for individuals. All entries are constructed as tables listing attributes followed by specific values.

4.2 Java Naming Directory Interface (JNDI)

JNDI is an API (Application Interface) specified in Java^{TM} that provides naming and directory functionality to applications written in Java. It is designed especially for Java by using Java's object model. Using JNDI, Java applications can store and retrieve named Java objects of any type. In addition, JNDI provides methods for performing standard directory operations, such as associating attributes with objects and searching for objects using their attributes.

JNDI is also defined independent of any specific naming or directory service implementation. It enables Java applications to access different, possibly multiple, naming and directory services using a common API. Different naming and directory service providers can be plugged in seamlessly behind this common API. This allows Java applications to take advantage of information in a variety of existing naming and directory services, such as LDAP, NDS (Novell Directory Services) [4], DNS (Domain Name Service) [5], and NIS (Network Information Service) [6], and allows Java applications to coexist with legacy applications and systems.

Using JNDI as a tool, the Java application developer can build new, powerful and portable applications that not only take advantage of Java's object model but are also well integrated with the environment in which they are deployed.

The computing environment of an enterprise typically consists of several naming facilities often representing different parts of a composite namespace. For e.g. an Internet Domain System may be used as the top level naming facility for different organizations within an enterprise. The organizations themselves may use a directory service such as

LDAP or NDS or NIS. From a user's perspective there is one namespace consisting of composite names.

The JNDI architecture consists of an API (Application Programming Interface) and an SPI (Service Provider Interface). Java applications use this API to access a variety of naming and directory services. The JNDI SPI provides the means by which different naming/directory service providers can develop and hook up their respective implementations so that the corresponding services are accessible from applications that use JNDI.

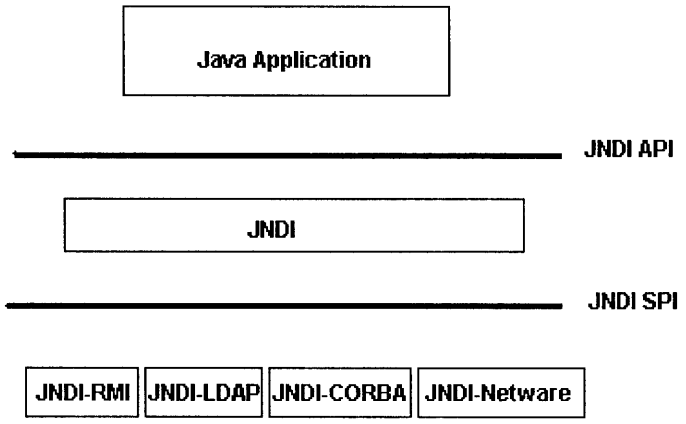


Figure 4-3: JNDI Architecture

In addition, because JNDI allows specification of names that span multiple namespaces, if one service provider implementation needs to interact with another in order to complete an operation, the SPI provides methods that allow different provider implementations to cooperate to complete client JNDI operations. This allows the user to navigate across several directory and naming services while working with seemingly only one logical namespace [7].

The next chapter describes the data exchange framework, which utilizes LDAP to locate web services that provide data conversion facilities.

5.0 InfoX ARCHITECTURE

Our goal is to design a *Data Exchange* architecture with the following attributes in mind.

- Scalability
- Performance
- Security
- Manageability.

Important issues to be decided on include how inter-system communication takes place and where the processing is done. We have at the least two common architectures available to evaluate.

5.1 Peer to Peer

A peer-to-peer architecture as depicted in Figure 5.1, is a truly distributed system. Examples include Sun's JXTA [8] implemented in Java and various implementations of the Gnutella [9] protocol. The primary advantage of such a system is the absence of a single point of failure. This advantage comes with an associated weakness, namely the absence of a central control point. From a pure security point of view, we would like to maintain a central checkpoint for all data exchange and control. Peer-to-Peer is still an evolving technology and has not yet fully matured. Although there are partially implemented peer-to-peer architectures such as Napster, it is still a long way before the infrastructure becomes fully available for implementing it in its pure form.

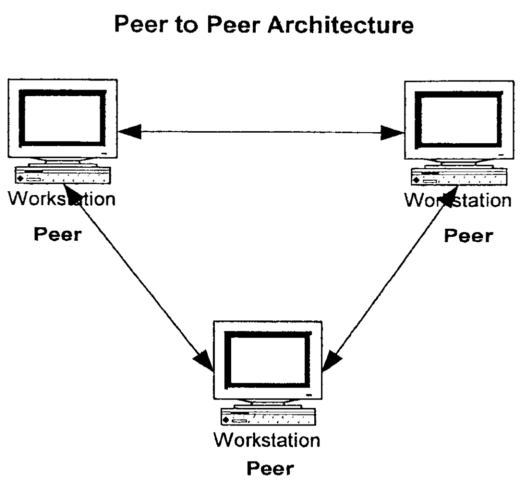


Figure 5-1: Peer-to-Peer Architecture

5.2 Hub and Spoke

The alternative to the first approach is a truly centralized system as depicted in Figure 5.2 below. We envision a large number of organizations using this system to exchange information, Hence the system needs to be scalable, safe and easily managed.

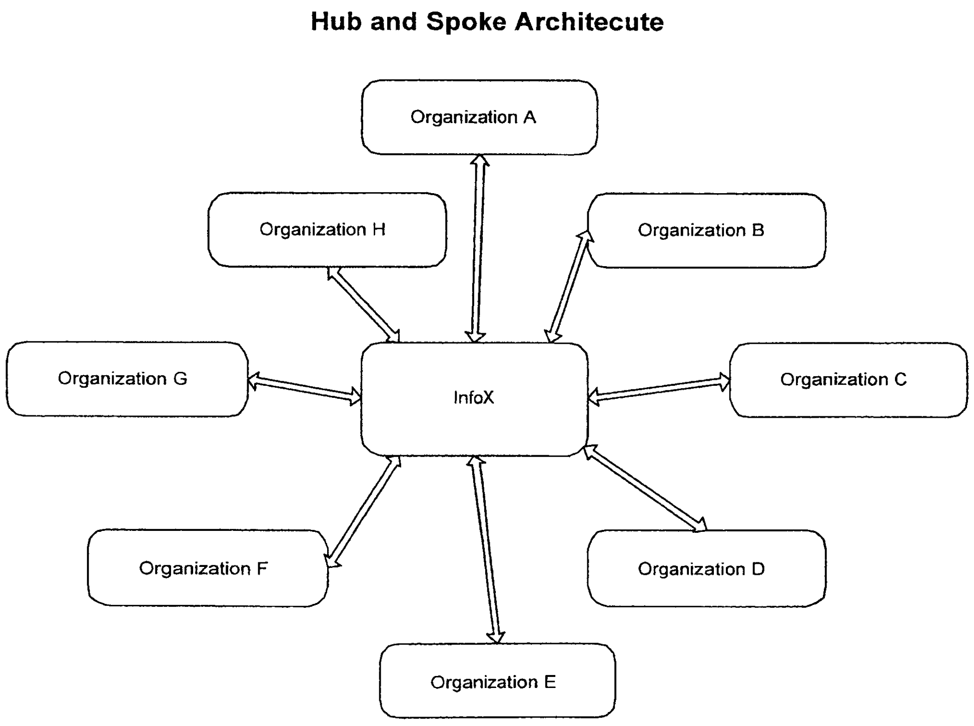


Figure 5-2: Hub and Spoke Architecture

A Hub and Spoke kind of architecture meets most of the requirements highlighted at the beginning of this chapter to a satisfactory level. Although a large number of organizations will participate in the exchange, the order of magnitude will be in tens of thousands as opposed to millions and therefore will meet scalability requirements. Since most of the information goes through a common server, there can be better controls established to provide the security required for data exchange between organizations. This architecture has the following added advantages, which cater for some of the other requirements such as manageability.

5.2.1 Tracking

The architecture provides a centralized location for logging and tracking. Carrying out reporting does not involve visiting the tracking databases of multiple machines – a coherent view is available from a single database.

5.2.2 Control

An emergency shutdown of the system can be accomplished quickly by bringing down just the hub. This can be useful in the event of a concentrated network attack or a fast spreading virus.

5.2.3 Filtering / Transformation

The architecture provides the basis for filtering or transforming files that travel through the hub. For example, if a hub-resident business process wishes to prevent a file from being forwarded on, it can do so. It might carry out such filtering based on keywords, type, virus checks, or other criteria.

5.2.4 Reduction of Interdependencies

Problems involving the availability of subsystems are not an issue in this kind of architecture. Any problem that arises can be easily isolated and that part of the system can be decoupled from the remaining system so that a failure in one system does not impede the operation of others.

5.2.5 Forensics

Rather than having to scrutinize every possible peer-to-peer pathway in the event of difficulties, operators have fewer possibilities to inspect.

5.3 Proposed Architecture

The proposed software architecture consists of five major modules as seen in Figure 5.3.

- **Security Infrastructure**
- **Transport (Communication) Layer**
- **Query Engine**
- **Rule Set Generation Engine**
- **Data Transformation Engine**

The security infrastraucture allows for proper authentication between client organizations to exchange data through InfoX server. The transportation layer accommodates various protocols and is responsible for delivery of data between the various modules on the client side and the InfoX server. The query engine module is responsible for sending and analyzing requests between the client and the InfoX architecuture. The rule set generation engine provides the guidelines on how the data transformation engine can transform data from one data type to another. These five modules are further described in detail in the following sections.

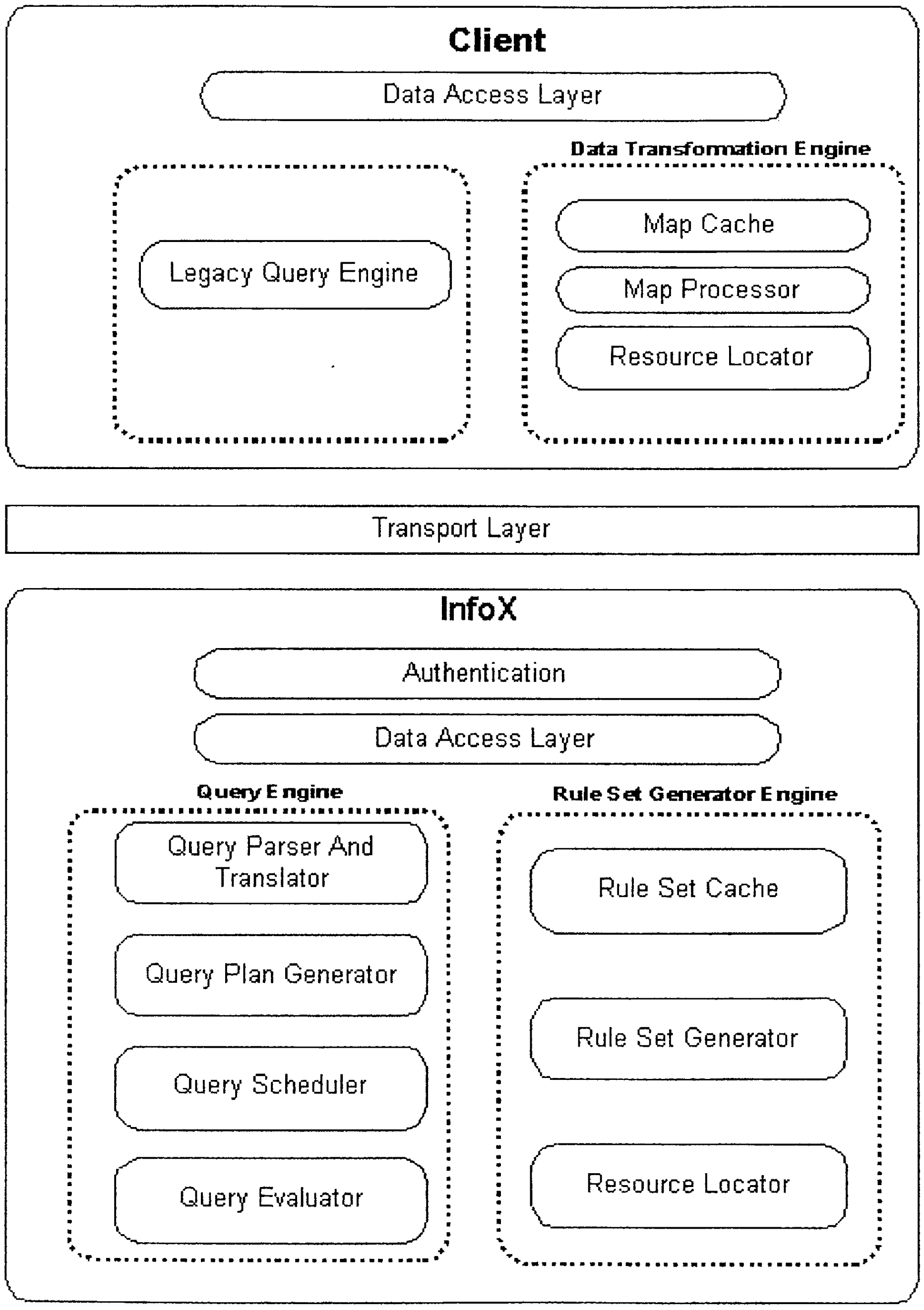


Figure 5-3: Data Exchange Architecture

5.3.1 *Security Infrastructure*

The data that will be exchanged through this system could be highly confidential and sensitive. Therefore, security may be of paramount importance to protect and ensure the integrity of the data being exchanged. The security system that will be used not only has to provide the best possible security but also has to scale to handle the huge volume of data that will be passing through the system. So it is crucial to choose the right kind of security infrastructure that should be used for this system.

All communication between different organizations that will be using this system will use the Transmission Control Protocol/Internet Protocol (TCP/IP) as the underlying protocol. TCP/IP allows information to be sent from one computer to another through a variety of intermediate computers and separate networks before it reaches its destination.

The flexibility TCP/IP offers makes it an ideal choice for the protocol to be used for all communications. TCP/IP is also the de-facto protocol that is used for all internet communications. So no additional infrastructure is necessary for the system to work. At the same time, the fact that TCP/IP allows information to pass through intermediate computers makes it possible for a third party to interfere with communications in the following ways:

- **Eavesdropping.** Information remains intact, but its privacy is compromised. For example, someone could learn your credit card number, record a sensitive conversation, or intercept classified information.

- **Tampering.** Information in transit is changed or replaced and then sent on to the recipient. For example, someone could alter an order for goods or change a person's resume.
- **Impersonation.** Information passes to a person who poses as the intended recipient. Impersonation can take two forms:
    - **Spoofing.** A person can pretend to be someone else. For example, a person can pretend to have the email address `foo@example.com`, or a computer can identify itself as a site called `www.example.com` when it is not. This type of impersonation is known as spoofing.
    - **Misrepresentation.** A person or organization can misrepresent itself. For example, suppose the site `www.example.com` pretends to be a furniture store when it is really just a site that takes credit-card payments but never sends any goods.

A set of well-established techniques and standards known as **public-key cryptography** [10] provides a solution to all the above issues. Compared with symmetric-key encryption, public-key encryption requires more computation and is therefore not always appropriate for large amounts of data. However, it's possible to use public-key encryption to send a symmetric key, which can then be used to encrypt additional data.

5.3.1.1 Message Authentication Code

A message authentication code (MAC) corresponds to a short and quickly generated/verified non-transferable signature on a document. Since it cannot be transferred (i.e., verified by a participant other than the one it was intended for) it cannot be used for contracts or receipts (if these need to be saved in case of a conflict.) but can be used for participants to make sure that the message they obtain is from the person they expect. Since they are very efficient, this makes them very useful for individual, small messages in interactive protocols. Here, all of these messages can later be signed if a receipt is needed. MACs require that the sender and the receiver of the authenticated message both know a (symmetric) secret that is used both for generating and verifying the MAC. This secret can be produced by one of the participants, and sent over in an encrypted form to the other, using a public key encryption method. MAC's can be implemented using stream ciphers, e.g., RC5 [11].

5.3.2 Transport Mechanisms

Because of the practical limitations and the infrastructure that is already in place at different organizations it becomes necessary to support a wide range of transport mechanisms.

The communication (application level) protocols that we have identified are:

- **SMTP**
- **A custom protocol using TCP/IP**
- **SOAP**
- **FTP**
- **HTTP**
- **XML/RPC**
- **Corba**

Currently SOAP is evolving to be the defacto standard and is soon replacing other communication transport mechanisms. SOAP is an open standard with a growing body of developers and vendors supporting it. As more vendors offer SOAP products and services, the advantages of using SOAP will become more pronounced. As outlined in chapter 3, there are many advantages that SOAP brings to the data exchange transport mechanism.

5.3.3 Query Engine

The Query engine presents a common, homogeneous language interface to access information present in the distributed system. It is the responsibility of the query engine to abstract out all the peculiarities and dissimilarities of the different systems and present a common interface to all the systems. It makes sense of and executes queries that come its way.

The different modules within the Query Engine are.

5.3.3.1 Query Parser and Translator

It is the duty of the Query Parser to translate all the requests for information that are sent to into an internal format (binary format that can be executed).

5.3.3.2 Query Plan Generator and Optimizer

The execution plan detailing the steps that have to be followed to generate the result is created by this module. In this process of generating the plan, it also optimizes the route that has to be followed to extract the information.

5.3.3.3 Query Evaluator

Once the optimized plan has been generated, it is the responsibility of this module to actually execute it and generate the result. Once the query has been executed and the required information extracted the binary information is converted back into an external format, which everyone understands.

5.3.3.4 Query Scheduler

Since this is a distributed system and also since the request for information could be made in an asynchronous manner an efficient scheduling algorithm is necessary. A scheduling system also ensures that critical requests for information are serviced first.

5.3.4 Rule Set Generation Engine

The rule set generation engine is the most crucial component in the whole system. It generates the rules according to which data is transformed in the system. This makes it possible for the different systems to talk to each other.

5.3.4.1 Maplets

The term maplet, which is uniquely defined for our architecture, is in its most general form, a binary executable, run in the Data Transformation Engine. The Rule Set Engine decides the maplets, which are necessary to perform the different transformations. A comprehensive description of maplets can be found in the section dealing with the data transformation engine.

5.3.4.2 Rule Set Generation Engine Implementation

This engine analyzes the sample response format (an XML document) for a query result. Depending on the namespace of the XML document that has to be transformed and by looking at the root node of the document, the rule set engine queries the directory for the appropriate maplet to do the transformation. If a matching maplet is found then it is added to the rule set. On the other hand, if a matching maplet is not found, it then looks at the next level of nodes and tries to find a maplet to transform these nodes, this process is repeated until the last individual element of the XML document is reached. The actual XML document generated as the response to the query will consist of repetitions of the above data elements. By repeatedly applying the rule set, the whole document can be transformed. The Rule Set Generator engine thus builds the rule-set required to transform the data to the common format.

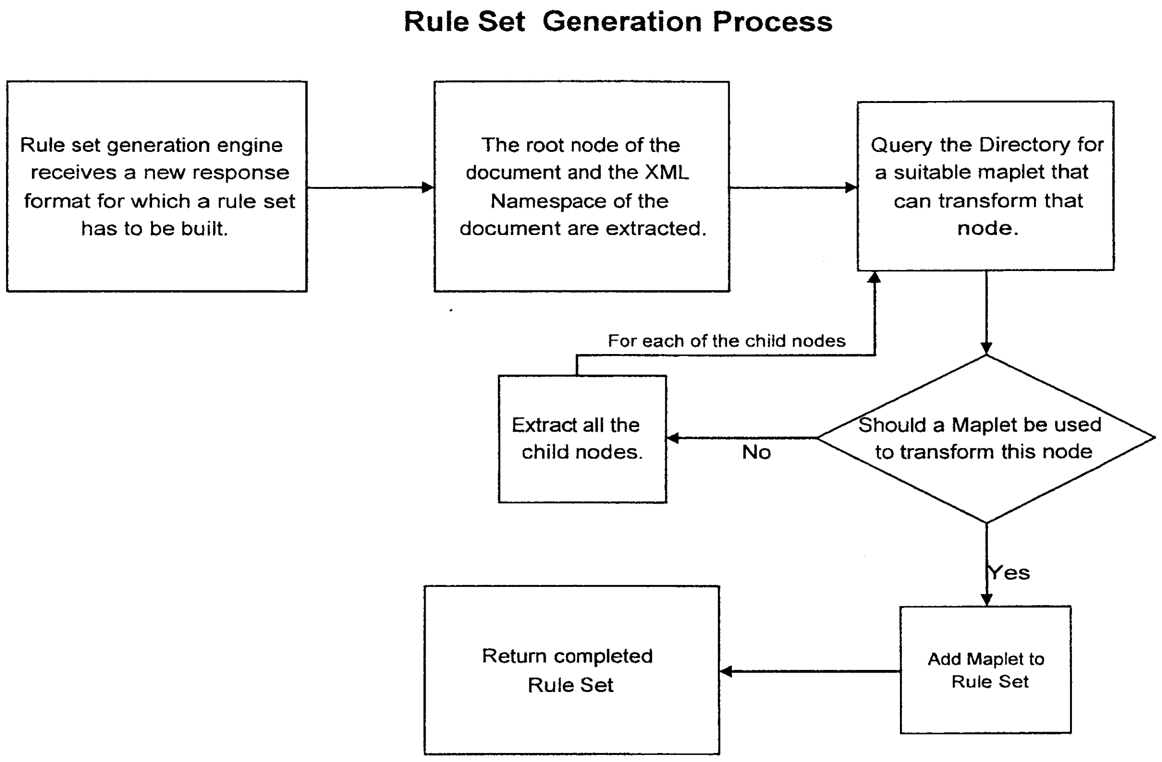

Rule Set Generation Process
Rule set generation engine receives a new response format for which a rule set has to be built.
The root node of the document and the XML Namespace of the document are extracted.
Query the Directory for a suitable maplet that can transform that node.
For each of the child nodes
Extract all the child nodes.
No
Should a Maplet be used to transform this node
Yes
Return completed Rule Set
Add Maplet to Rule Set

Figure 5-4: **Rule Set Generation Process**

The rule set generator engine has to build two rule sets. Let the query originate from client A. Let client B provide the answer. One rule set transforms the query results from B to the common data format. Another rule set transforms the data from the common format to a format that client A understands. The rule set generator engine just builds the rule sets. The actual processing of the information takes place on the client side. This keeps the system scalable, as the rule set generator engine does not waste its resources in running continuous transformations, but just plays a supervisory role.

5.3.5 Data Transformation Engine

Data transformation engine does the actual transformation of data according to the rule sets built by the rule set generator. Here we use the concept of Data Map. A Map contains information on how to convert data in one format into another format. An example of a Map could be a XSLT sheet that transforms an XML document into another XML document. The components of the engine are

- MAP Processor
- MAP Cache
- Resource Locator
- Data Access API

5.3.5.1 MAP Processor

This is the module which actually transforms data across different Models.

5.3.5.2 MAP Cache

The Map processor uses different Maps to do the data transformations. The maps that are used can be loaded as required. But before they can be used, they have to be compiled, which is time consuming. Since the system has to be designed for high volumes, caching all these objects in memory can have tremendous performance boosts.

5.3.5.3 Resource Locator

During the process of applying the MAPS to the data a number of resources have to be used and different services also may have to be utilized. So the MAP processor uses the resource locator to access the different resources that have to be used in the process of data transformation.

5.3.5.4 Data Access API

This layer provides a uniform and consistent way of accessing data from disparate data sources. The different data sources could involve Relational Databases, LDAP compliant Directory Services or Legacy Data Stores.

5.3.5.5 Data Transformation Engine Implementation

5.3.5.5.1 Maplets

A maplet is defined as a component of the system, which offers Data Transformation Service from one data model to another data model. Enforcing a common scheme on all the data transformation services provides a simple and consistent mechanism, which allows new collaborative applications to be developed and deployed. It also makes the management of these services very easy for the collaboration server.

A maplet lifecycle involves the following processes (Figure 5.5):

- The maplet is loaded into the Data Transformation engine and initialized.
- The maplet is used by the Data Transformation engine to handle one or more than one data transformation requests.
- When not in use, the maplet can be cached for future use.
- Maplet is unregistered from the Data Transformation Engine and is stopped.
- Typically, each Maplet will have its own set of resources using which it can provide the Data transformation services. Depending on the demand for a particular transformation service, the data transformation can have more than one instance of the maplet active at any given time.

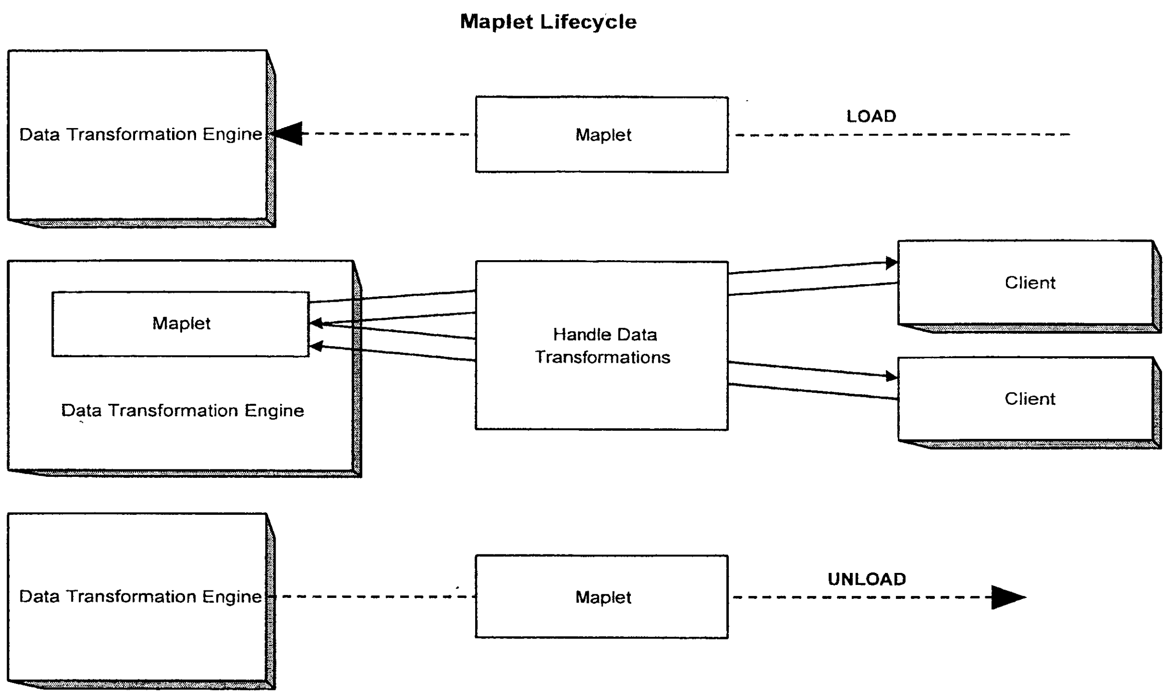

Maplet Lifecycle
Data Transformation Engine
Maplet
LOAD
Maplet
Data Transformation Engine
Handle Data Transformations
Client
Client
Data Transformation Engine
Maplet
UNLOAD

Figure 5-5: Maplet Lifecycle

5.3.5.5.2 Maplet Implementation Using XSLTs

XSLT presents a very elegant, extensible and convenient way of creating Maplets. All the data transformation rules and processing can be expressed in an XSLT. So every maplet will have an XSLT associated with it. For any XML input the maplet receives, it applies the XSLT on it and returns the new XML document that was produced from it.

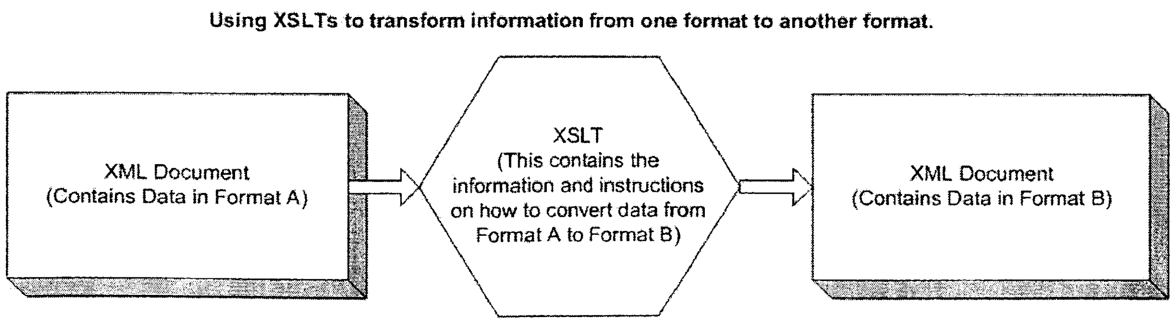

Using XSLTs to transform information from one format to another format.
XML Document
(Contains Data in Format A)
XSLT
(This contains the information and instructions on how to convert data from Format A to Format B)
XML Document
(Contains Data in Format B)

Figure 5-6: Using XSLT to transform information from one format to another format

The data transformation from one model to another can also involve complex processing. For example, in one data model temperature could be measured in Kelvin and in another model, the temperature could be measured in Fahrenheit. To transform temperature data across these data models, one has to convert data from Kelvin to Fahrenheit. This is an atomic transformation ie transformation of the leaf node data type. This functionality could be provided as a Web Service over by the internet by one or more organizations. As part of the transformation, it might be necessary to utilize the services of one or more of the Web Services.

Every organization's data model can be split into certain atomic entities. There could be more complex entities, which consist of one or more than one atomic entity. So any transformation that we do on the data model is comprised of transformation operations on the atomic entities. The process of data transformation would have been much simpler if there was a one to one mapping between these atomic entities across different data models. But this is hardly every the case. An atomic entity in one data model could be mapped to more than one atomic entity in another data model. To handle these cases we have two different kinds of maplets.

5.3.5.6 Simple Maplets

A simple maplet handles data transformations of the atomic entities in a data model. So for every atomic entity in a data model there will be a corresponding Simple Maplet, which transforms it into another Data Model.

5.3.5.7 Complex Maplets

A complex maplet deals with the transformation of more than one atomic entity. In doing so it could use more than one simple maplet or do it all by itself.

Example:

Consider the data model for a product order. The atomic entities in an Order are Product, Shipping Address and Billing Address. Therefore, a maplet, which transforms an Order across different data models, is a complex maplet and in doing the transformation, it could use the simple maplets for Product and Address.

5.3.6 Repository for Mapping Information

Since we will be dealing with an enormous amounts of information that will be used in the process of applying these maps, how we store the mapping information is of paramount importance. The first and obvious choice is to use a relational database management system (RDBMS). Another serious contender to this choice is a Directory. In the following table, we evaluate the pro and cons of these choices.

Directory	**Relational Database**
Data that is read frequently but updated much less frequently. Examples: names, addresses, phone numbers, passwords, interest profiles.	Data that is updated frequently. Examples: transactions, account balances, shopping cart contents.
Data that lends itself to hierarchical organization. Examples: names in an enterprise organization, customers in geographical regions.	Data that has complex relationships. Examples: sales orders, work orders.
Data that is general-purpose, and tends to be used in many disparate systems or that may turn out to be useful to future applications. Examples: names, addresses, phone numbers, passwords, interest	Data that is of use solely to a specific application. Example: salary data in the HR system; transactions and balances specific to a particular system.

profiles, locations, reporting structure.	
Data that is required at a variety of sites. Example: User profile information for a user who moves between computers.	Data that is needed for a particular program executed at a particular site. Example: site specific configuration information.
Data that is required for quick read access and where network considerations inhibit access to a central data store. Example: data required at remote offices that are connected via a slow link.	Data that can be obtained sufficiently rapidly from a centralized data store. Example: Really dependent on the network bandwidth, database retrieval performance and ability to bi-directionally replicate database data.
Data that can tolerate a latency in accuracy. Example: Credit limit	Data that requires real-time update for accuracy. Example: Current credit extended to date.

After considering the pros and cons of Databases and a Directory Server for storing the mapping information, it turns out that the Directory is a better way of storing the mapping information. There are many reasons for this but the primary one being that a directory is optimized for read only (due to less frequency updates) whereas the database is optimized for both read and write. This will save a significant amount of overhead thus allowing quick read access. In addition, directories can easily be exposed to the internet without much security risk while it is difficult to expose a whole database to the internet.

The following information is stored in the Directory.

1. Authentication Information.
2. Maplet Information, i.e., which maplet to use for data transformation from one model to another model.
3. A directory of all the web services that are being offered by different organizations/systems that can be used in the data transformation process.

In the previous chapter, significant detail has been provided on directory services such as LDAP, which InfoX hopes to use extensively to obtain the required data conversions sources.

5.4 Case Scenario

5.4.1 Problem Statement

An example scenario in a hypothetical business situation is described below.

We have three companies in this scenario. Company **X** makes a widget **W** that is sold all over the country. A maintains a database where it tracks the number of widgets it supplies to its distributors, superstores **Y** and **Z**. **X** would like to change its advertising policy for the widget by targeting states where sales of W is less than its competitor made by another company. For this, it needs regional sales data for its widget. Unfortunately, it does not have this information, but superstores **Y** and **Z** surely will. **X** decides to ask **Y** and **Z** for the relevant data. **Y** and **Z** maintain their own sales data. **Y** stores point of sales data, i.e., number of widgets sold per zip code. **Z** is a much bigger chain and partitions out sales data according to states where it has franchisees. **Y** and **Z** are competitors, so while they agree to give **X** the data it needs, they would like their data to be kept secret from each other.

5.4.2 InfoX Solution:

The InfoX Architecture Engine works as follows (see Figure 5.7 for the flow chart)

1. Client **X** would make a query request to InfoX Engine

2. The Query Engine of InfoX will parse the query, generate a plan, and evaluate it. Here, the query engine will identify the data source (client **Y**) from which it needs to obtain the requested information.
3. Client **Y** will receive the query request.
4. The query engine of Client **Y** will execute the query and send feedback to InfoX on the response format of the query request.
5. InfoX Rule Set Generator analyses this feed back and begins to generate rule sets which are explained below:
    - Rule set **Y** takes Client **Y**'s data and transforms it to the common data format. A special maplet which is obtained from a webservice located through a directory service is used to transform the zip codes to their respective states. The common data format encapsulates number widgets sold per zip code.
    - Rule set **X** is generated by the rule set generator to transform data from the common format to the format required by Client **X**'s data store.
6. The rule set **Y** is then applied by the Data Transformation engine of Client **Y** to the results obtained by the query engine in step 4 above. The InfoX rule set generator engine will set up the necessary infrastructure needed for securing this transaction. The data transfer is then monitored as client **Y** processes the data into the common format and sends it across to **X** for it to decode. The outcome will create the data in a common data format on which, Client **X** data transformation engine can now apply the rule set **X** on and convert it into a Client **X** required format.

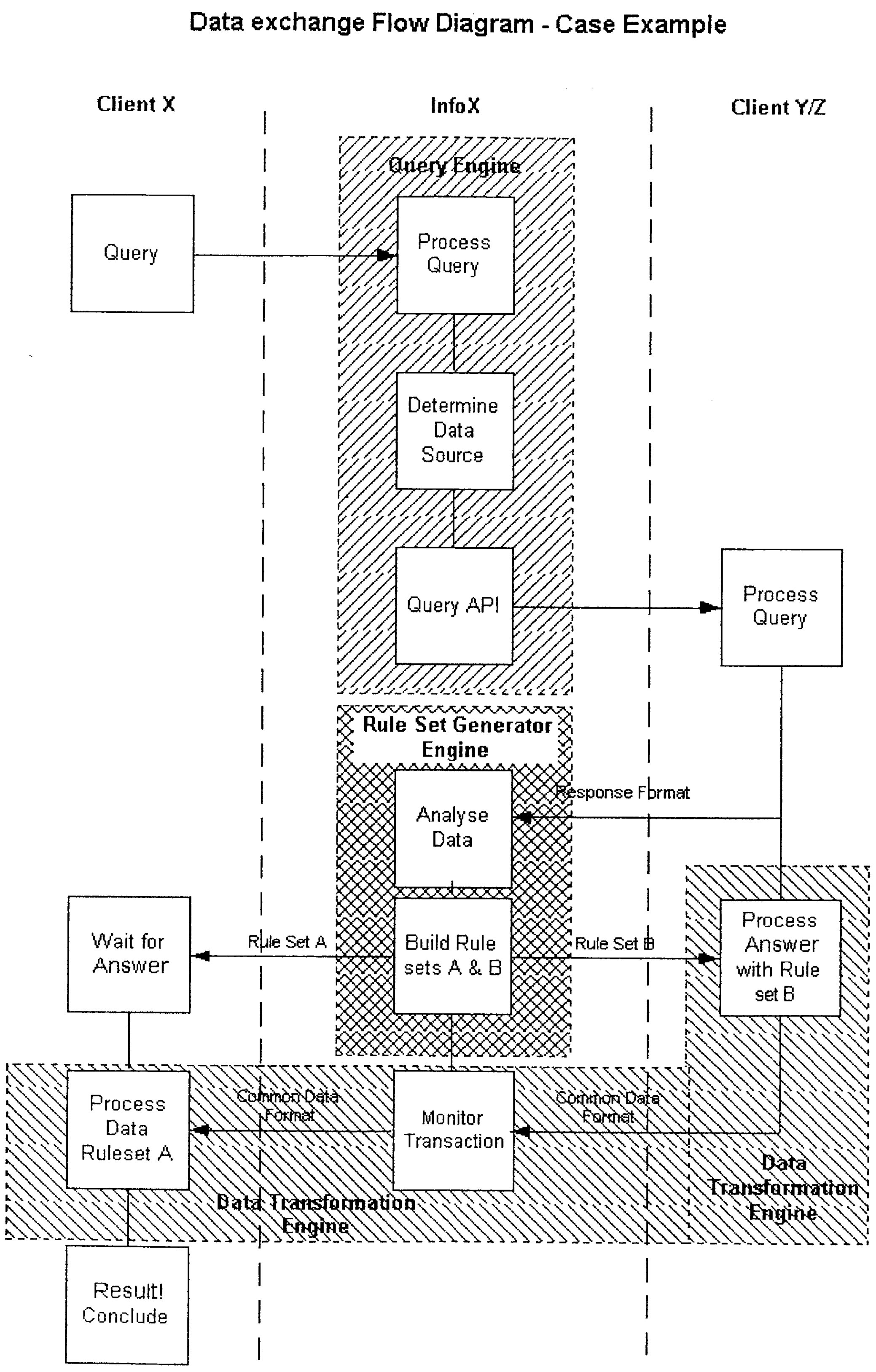


Figure 5-7: **Data Exchange Flow Diagram – Case Example**

Similarly, the process is repeated for Client **Z** to obtain the required information. Only now, Rule set **Z** is easier. Client **Z** returns number of widgets sold per state instead of by zip code.

5.5 Conclusion

As was set out at the beginning of this thesis work, the proposed Data Exchange architecture should have all the important attributes such as scalability, performance, security and manageability. These attributes were addressed in the following ways:

Scalability: This has been addressed in many ways during the design. For example, we have designed the rule set generator engine to build only the rule sets. The actual processing of the information will takes place on the client side to keep the system scalable, so that the rule set generator engine does not waste its resources in running continuous transformations. The concept of a common data format is also of importance. This allows the system to scale. If the transforms were done directly between clients, we would need O (N*N) rule sets to transform successfully among a total of N clients. The use of the common transform language cuts this number down to O (N).

Security: This has been addressed at various levels of the data exchange process by using well-established techniques like public-key cryptography.

Manageability: By using a truly centralized system, a large number of organizations can use this hub and spoke architecture system to exchange information without many interdependencies and hence is easily manageable. In addition, any changes in common data formats will not lead to an intensive over haul of the data model on the client side thus making it easy to manage this data exchange process over long periods during which technology and standards are bound to evolve.

Performance: We have identified areas within the architecture where performance issues needed to be addressed. For example, the Map processor uses different Maps to do the data transformations and need be loaded as required. This can be a time consuming exercise, as each map may need to be compiled. Since the system was designed for high volumes, caching all these objects in memory could give tremendous performance boosts. The same could be said for maplets when being loaded into the data transformation engine. When not in use the maplet could be cached for future use to speed up the process.

The next chapter will highlight the issues that were set out at the beginning and how the proposed architecture has addressed those issues. It also gives a brief insight on the directions for future research and development of the architecture.

6.0 CONCLUSION

With recent globalization trends, organizations increasingly have to exchange information between each other. The Internet is frequently being used as a global infrastructure for data exchange between autonomous participants. The complexity of this exchange process magnifies when an organization has to interface with others for B2B communication. One of the biggest challenges facing organizations today is integrating the multitude of different information systems that have been implemented over the years. The problem with these kinds of inter-organizational data exchanges is that they involve a large number of information systems, which do not necessarily share a consistent data model. They require the ability to exchange semi-structured data through some channel.

One approach is for every organization to conform to a standard, common and consistent data model. Most of the industry-sponsored initiatives take the approach. Despite a number failures and partial successes, this architecture will continue to be proposed. The reason is simple – typically in any asset intensive industry like automobiles, airlines etc. we see the presence of a few major players and a number of minor players. The major players have the capability to dictate, enforce and commit resources to such industry-sponsored initiatives because of the benefits accrued to them. This puts the CIOs of

smaller, resource-constrained organizations in a dilemma. CIO's are tasked with developing a computing and connectivity infrastructure that needs to keep pace with industry sponsored initiatives in a resource-constrained environment.
Should they integrate their applications with the currently proposed industry standard? What will happen to investments in integration when the standards change or revise? We define this risk as standards oriented risk. What happens when newer technologies gain momentum? We call this technology oriented risk. They need to reintegrate.

Should they choose they do nothing? In this case, they lose out to players that are more adept. This approach too is very risky and uncertain, because the CIO has chosen not to deal with standards and technology oriented risk. So what principle should dictate CIO's integration strategy? Any proposed architecture that reduces the standards risk and technology risk, and yet solves the problem of B2B communication, is therefore a potentially a superior architecture from the perspective of the CIO.

This thesis discussed the use of XML technologies for mapping information between partially consistent data models. The role of XML in semi-structured data exchange [12] is described together with its application as a framework for data exchange. A description of an XSLT based architecture, which will take the unshared XML schema elements of these data models and map them, is outlined. A directory service that provides for the location of a suitable conversion resource such as XML-RPC / SOAP (a brief overview of which is included in this thesis) for satisfying the second stage of the discovery process is also described.

The architecture also deals positively with issues such as scalability, manageability, performance and security, which are, are key components for any practical implementation to be successful.

However, this thesis is built around an idea. The idea that XML technologies will become fully evolved, available and flexible with time. Within the architecture described, the query engine needs to be researched on further. There are many query languages out in the market but recently, XML query has been gaining popularity and may well be part of the solution we have been looking for. Chapter 5 dealt with the mapping and data conversion issues using XSLT and directory services. There was not much emphasis on the design of the query engine, which consists of the query parser, plan generator, scheduler and evaluator.

The work presented here, is therefore a long way from a stage when the proposed architecture could be implemented as it is and produce very robust results. However, it has laid a foundation for further research and taken a step in the right direction. As I began to scratch the surface of this project, I quickly learned that there were many areas that needed detailed research than could possible be included in a single Master's project. If the work continues, along the lines suggested above and some of these technologies keep evolving, this architecture will prove extremely valuable to all those organizations caught up in the globalization frenzy.

7.0 BIBLIOGRAPHY

[1] "*Namespaces in XML*", W3C Recommendation 14 January 1999, *http://www.w3.org/TR/1999/REC-xml-names-19990114/.*

[2] Mark Reinhold, "An *XML Data Binding Facility for the Java Platform*", White Paper July 1999, *http://java.sun.com/xml/docs/bind.pdf*.

[3] Cellucci, Joseph; Hill Russell; Simon. Alan. -- *"You Are Here -- New developments in directory services have managers wondering which way to take their corporate networks."* -- Communications Week. -- November 11, 1996, issue 637. -- Section: CloseUp -- Directory Services, *http://www.techweb.com/se/directlink.cgi?*

[4] Novell Homepage, *"The NDS Development Environment"*, White Paper, *http://developer.novell.com/whitepapers/nds1.htm*

[5] Nicolai Langfeldt, *"The Concise Guide to DNS and BIND"*, Que, first edition, November 2000.

[6] Thorsten Kukuk, *"The Linux NIS/ NIS+ HOWTO"*, White Paper, November 2000, *http://www.linux-nis.org/nis-howto/HOWTO/index.html*

[7] Richard Monson-Haefel, *"The Java Naming and Directory Interface (JNDI): A More Open and Flexible Model"*, *http://www.javareport.com/html/features/archive/9802/haefel.shtml*

[8] Li Gong, *"Project JXTA: A Technology Overview"*, White Paper, April 2001, *http://www.jxta.org/project/www/white_papers.html*

[9] Nelson Minar, "*Distributed Systems Topologies: Part1*" December 2001, *http://www.openp2p.com/pub/a/p2p/2001/12/14/topologies_one.html*

[10] Burt Kaliski, "*Standard Specifications for Public-Key Cryptography*", August 1999, *http://grouper.ieee.org/groups/1363/P1363/presentation.html*

[11] Johan Borst, *"Linear Cryptanalysis of RC5 and RC6",* 1999, *http://citeseer.nj.nec.com/248955.html*

[12] Eswar Vemulapalli, *"The Use of XML Technologies for Mapping Information between Partially Consistent Data Models"* ATRIP Conference, April 2001.

8.0 Appendix A

Sample SOAP Code

In this appendix, a complete example of requests and responses written in SOAP messages are provided.

The following request is an example of a SOAP message embedded in a HTTP request. Organization A is requesting the namespace identification of Organization B.

```
POST /Definition HTTP/1.1
Host: 209.110.197.12
Content-Type: text/xml;
Content-Length: 162
SOAPMethodName: “URL”#getidentification

<se:Envelope
  xmlns: se="http://schemas.xmlsoap.org/soap/envelope/"
  se:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
   <se:Body>
      <m:getidentification xmlns:m="URL">
                  <org>OrgB</org>
      </m:getidentification>
   </se:Body>
```

```
</se:Envelope>
```

Following is the response message from OrgB, containing the HTTP message with the SOAP message as the payload.

```
HTTP/1.1 200 OK
Content-Type: text/xml;
Content-Length: 162

<se:Envelope
  xmlns:se="http://schemas.xmlsoap.org/soap/envelope/"
  se:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
   <se:Body>
      <m:getidentificationResponse xmlns:m="URL">
         <result>url//xxxx</result>
      </m:getidentificationResponse>
   </se:Body>
</se:Envelope>
```

The following SOAP message is sent to Org B requesting for the details of the object Destroyer

```
POST /Definition HTTP/1.1
Host: 209.110.197.12
Content-Type: text/xml;
Content-Length: 162
SOAPMethodName: "URL"#getobjectdetails

<se:Envelope
  xmlns: se="http://schemas.xmlsoap.org/soap/envelope/"
  se:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
   <se:Body>
```

```
      <m:getobjectdetails xmlns:m="URL">
                  <objID>5678</objID>
      </m:getobjectdetails>
  </se:Body>
</se:Envelope>
```

Following is the response message from OrgB, containing the HTTP message with the SOAP message giving the schema identification of the object destroyer.

```
HTTP/1.1 200 OK
Content-Type: text/xml;
Content-Length: 162

<se:Envelope
  xmlns:se="http://schemas.xmlsoap.org/soap/envelope/"
  se:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
   <se:Body>
      <m:getobjectdetailsResponse xmlns:m="URL">
         <objname>url//xxxx</objname>
         <schema>yyyyy</schema>
      </m:getobjectdetailsResponse>
   </se:Body>
</se:Envelope>
```