2007

XML Integrated Environment for Service-Oriented Data Management

Marwan Younes Maarouf
Wright State University

XML INTEGRATED ENVIRONMENT FOR SERVICE-ORIENTED DATA MANAGEMENT

A dissertation submitted in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy

By

MARWAN YOUNES MAAROUF
M.S., Wright State University, 1991
B.S., Youngstown State University, 1988

2007
Wright State University

WRIGHT STATE UNIVERSITY

SCHOOL OF GRADUATE STUDIES

May, 16 2007

I HEREBY RECOMMEND THAT THE DISSERTATION PREPARED UNDER MY SUPERVISION BY Marwan Younes Maarouf ENTITLED XML Integrated Environment For Service-Oriented Data Management BE ACCEPTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF Doctor of Philosophy.

Soon M. Chung, Ph.D.
Dissertation Director

Thomas Sudkamp, Ph.D.
Ph.D. Program Director of
Computer Science and Engineering

Joseph F. Thomas, Jr., Ph.D.
Dean, School of Graduate Studies

Committee on Final Examination

Soon M. Chung, Ph.D.

Krishnaprasad Thirunarayan, Ph.D.

Thomas Hartrum, Ph.D.

Raymond Hill, Ph.D.

Michael Talbert, Ph.D.

ABSTRACT

Maarouf, Marwan Younes. Ph.D., Department of Computer Science and Engineering, Wright State University, 2007. XML Integrated Environment for Service-Oriented Data Management.

The proliferation of XML as a family of related standards including a markup language (XML), formatting semantics (XSL style sheets), a linking syntax (XLINK), and appropriate data schema standards have emerged as a de facto standard for encoding and sharing data between various applications. XML is designed to be simple, easily parsed and self-describing. XML is based on and support the idea of separation of concerns: information content is separated from information rendering, and relationships between data elements are provided via simple nesting and references. As the XML content grows, the ability to handle schemaless XML documents becomes more critical as most XML documents do not have schema or Document Type Definitions (DTDs). In addition, XML content and XML tools are often required to be combined in effective ways for better performance and higher flexibility. In this research, we proposed XML Integrated Environment (XIE) which is a general-purpose service-oriented architecture for processing XML documents in a scalable and efficient fashion. The XIE supports a new software service model that provides a proper abstraction to describe a service and divide it into four components: structure, connection, interface and logic. We also proposed and implemented XIE Service Language (XIESL) that can capture the creation and maintenance of the XML processes and the data flow specified by the user and then orchestrates the interactions between different XIE services. Moreover, XIESL manages the complexity of XML processing by implementing an XML processing pipeline that

enables better management, control, interpretation and presentation of the XML data even for non-professional users. The XML Integrated Environment is envisioned to revolutionize the way non-professional programmers see, work and manage their XML assets. It offers them powerful tools and constructs to fully utilize the XML processing power embedded in its unified framework and service-oriented architecture.

TABLE OF CONTENTS

LIST OF FIGURES

ACKNOWLEDGEMENT

I would like to express my deep and sincere gratitude to my supervisor, Professor Soon M. Chung. His support, encouragement and personal guidance have provided me with an outstanding foundation to build on during this research.

Besides my advisor, I wish to express my warm and sincere thanks to the rest of my dissertation committee: Professor T.K. Prasad, Professor Thomas Hartrum, Professor Raymond Hill and Professor Michael Talbert. With their insightful comments and questions, they contributed a lot to this work and kept me focused on what is important.

This dissertation concludes an exciting and challenging goal that I had set for myself over 5 years ago. However, it is only a starting point of what I believe and pray to be my continued effort in pursuing good and useful research that serves science and humanity.

Maarouf, Marwan Y.

May 2007

Dayton, Ohio, USA

DEDICATION

To my parents, wife and children for their continued prayers, love, encouragement and support.

Chapter 1

Introduction

1.1 Internet and XML Computing

The explosive growth and spread of the internet and its technologies have brought revolutionary changes, opened new horizons, and raised the bar very high for future computing systems and applications. In order to meet the new challenges, more interactive, flexible and scalable distributed heterogeneous infrastructure services are developed and rapidly deployed. Enterprise platforms like the Java Enterprise Edition (JEE) are becoming the environment choice for developing and deploying enterprise applications. JEE consists of a set of services, application programming interfaces (APIs), and protocols that provide the functionality for developing multi-tiered, Web-based applications [11].

At the heart of this revolution is the data that is transferred, displayed, translated, parsed, stored, searched, indexed, and always changing and expanding. Users are demanding simple, flexible, consistent and reliable services on the World Wide Web (WWW) regardless of the complexity of the data, the uncertainty in the infrastructure, or the shortage of the underlying distributed resources [12].

Extensible Markup Language (XML) is being used to deliver more richly structured content over the web and other important platforms. It is extensible because it does not have a fixed format, unlike HTML. XML is not a single, predefined markup language, rather it is a meta-language (a language for describing other languages), which allows us to design our own mark-up [1].

A predefined markup language like HTML defines a way to describe information in one specific class of documents. XML, on the other hand, lets you define your own customized markup languages for different classes of documents.

The World Wide Web Consortium (W3C) calls XML "a common syntax for expressing structure in data" [1]. Structured data refers to data that is tagged for its content, meaning, or use. XML is a public format: it is not the proprietary product of any company. XML allows groups of people or organizations to create their own customized markup applications for exchanging information in their domain. XML is called the "global currency" for the information economy [29]. Some of its key attributes include:

- Describing of data in a human-readable format. In addition, XML can be easily interpreted and processed by a computer.
- XML documents are self-describing. XML is a meta language for describing data in the form of tags and attributes.
- Separation of content and presentation – XML tags describe meaning. The presentation or display properties can be controlled by XSL style sheets, which provides a way to separate display properties from the data and where the same content of the document can be displayed in multiple places using different formats.

- Additional XML tags can be created easily and as required. In another words, XML does not have a fixed set of tags. XML is a meta-markup language. It allows us to define our own markup languages (i.e., our own vocabulary, our own grammar).
- Flexible and efficient data handling – any required data type is possible, and existing data structures can be mapped to XML. In searching data, XML reduces search time substantially, results in more accurate searches, and enables more effective searches.
- A standard data and document interchange format independent of platform, language, and vendor.
- Author identification and versioning at the element level – any XML tag can include the attributes that are needed to identify author or version information.
- Supporting of multilingual documents – XML documents are written in Unicode which can contain characters from practically all known written languages. Unicode support is particularly useful for web services.

Today, XML is playing a dominant role as a data interchange format in business-to-business (B2B) Web applications, such as e-commerce, supply-chain management, workflow, and application integration. XML is used in Web services as a platform-independent method for creating messages. XML supports network efficiency by reducing redundant data flow, eliminating the need to retrieve entire records, and allowing posting new data without refreshing the page [9].

XML is also useful for structured information management, including information contained in databases. XML supports media-independent publishing by allowing documents to be written once and published in multiple media formats and devices. XML enables a single view of all data regardless of database organization. It supports rich, hierarchical, structured and semi-structured forms of data [7].

XML is becoming the dominant format for internet-based application-to-application interactions. Web servers and related applications can easily and quickly build their own information in a common syntax and exchange it easily, thus allowing better and easier interoperability between the information providers. In fact, XML dominance as an interoperability-enabling technology will ensure that these interactions will grow considerably faster than application-to-person interactions.

1.2 Motivation

Today's XML tools like XML parsers, XSLT transformers, XML validators and Web Services do a good job in storing, parsing, transforming, exchanging and presenting XML content. In addition, countless XML applications are being developed to address different XML specific requirements like formatting XML data for output to screen, paper or other media (XSLFO) and querying XML data including databases (XQuery). XML tools often require to be combined. For example, validating the resulting document after an XSLT transformation; and updating a database after querying and manipulating an XML info set. Thus, there is a need for effective ways and mechanisms of combining XML content and XML tools for better performance and higher flexibility. We can also consider this as an integration challenge for all these technologies. For instance,

application integration requires the synchronization and coordination of XML data processing flows, in addition to the possible orchestration and automation of business processes, due to e-business trends such as virtual supply chains and business-to-business commerce.

XML processing usually involves multiple operations where branches, conditions and validations also play a major role. So, managing the complexity of XML processing by using processing pipelines allows the XML processing to be composed of a number of smaller, simpler components. Moreover, it allows unit testing and reduces the effort, cost and risk of integration.

To illustrate the difficulty of dealing with today's tools, we present a simple XSLT transformer example that only touches the surface of the complexity of developing a solution for even one of the simplest XML tasks – XML transformation.

XSLT is a language for transforming XML documents into other XML documents. XSLT is designed for use as part of XSL, which is a stylesheet language for XML. Typically, the transformation is done by implementing an XSLT transformer with two inputs and one output, all of which are XML documents as depicted in Figure 1:

Figure 1: XSLT Transformer

- An input document to transform (XML-Doc)
- An input style-sheet document that specifies the transformation (XSLT-Doc)

- An output transformed document (XML-Transformed-Doc)

There are two primary methods of implementing such a transformation. The first method is to use a command-line transformer that will read and parse the input documents (XML-DOC and XSLT-DOC) and output the result as files (XML-Transformed-Doc). This solution is efficient when a single transformation is required and the data is available in the form of files. However, it is poorly sensitive to situations where high-performance is important. It requires parsing of the files every time a transformation is performed and does not support optimizations techniques such as caching stylesheets, inputs and/or valuable intermediate results.

The second method is to develop an application that will perform the transformation, for example, by using JAXP (Java API for XML Processing). This solution is more flexible but requires an extensive knowledge of XML APIs. Java programming skills are needed to use JAXP to read and parse the files needed for a transformation. In case of cascading XSLT transformations, a SAX pipeline is required to process the XML elements on the fly, or a DOM tree can be used to store the intermediate results for further processing. The learning curve for a low-level API-based solution is steep and complex and out of reach for non-programmers. While many programmers utilize low-level APIs like JAXP for simple XML processing or a single-shot XSLT transformation, going further to construct processing pipelines often proves difficult especially when advanced features such as conditionals, validations, caching and debugging are desired.

XML continues to be rapidly accepted, adopted and augmented with parsing, processing, and style sheets together with other related standards to bring it up to

"enterprise strength" as an integrated standard [9]. In the midst of this technical revolution, we also seem to forget a simple fact that most of the people who work with the data are non-professional programmers. The people who generate, modify, publish and work with the data are non-experts and don't have the computer skills to handle the complex tasks at hand without the help of expert programmers or powerful applications. Recent history tells us that HTML, a simple markup language that can be used by non-experts, revolutionized electronic publishing by enabling non-professional programmers to distribute information on the Internet in electronic form. Our proposed XML Integrated Environment (XIE) is envisioned to revolutionize the way non-professional programmers see, work and manage their XML assets. It provides tools and constructs to fully utilize the XML processing power embedded in its unified framework and service-oriented architecture.

1.3 Usage of XIE

With XIE, we can quickly build, execute and debug simple and complex schemaless XML data flows. The data flows can track and profile user activity to support process improvement plans and strategies. For example, it is possible to learn and create XML vocabularies and tools to detect and resolve discrepancies among XML tags by building specialized XML processing services with the help of built-in XML transformation and XML updates services, and hence facilitating the integration of information. Also, the XIE can serve as a dynamic integration and incremental server allowing better control and easier debugging of the XML services that are executed within it. In addition, the XIE can be used as a candidate environment for filtering,

classifying and routing the growing number of XML data documents associated with large-scale information dissemination systems and XML messages associated with web services [5, 8].

Moreover, with XIE, we can quickly create and efficiently execute very specific XML computing tasks. Some use cases include but not limited to the following [4]:

1) Style an XML document in a browser with one of several different style sheets without having multiple copies of the document containing different XML style sheet directives.
2) Style the different elements of an XML document with the same style sheet without having to recompile it again for each element.
3) Apply a sequence of operations such as routing, validation, and transformation to a document. If the result or an intermediate stage is not valid, it can be aborted.
4) Allow an application on a handheld device to construct a pipeline, send the pipeline and some data to the server, allow the server to process the pipeline and send the result back.
5) Process large XML documents: Processing large (300MB or larger) XML files to extract particular element(s) that need processing (e.g. transformation) which are repeated over-and-over. An XPath step can be applied in a streaming fashion to the input. The matching info sets (i.e. the particular elements) are produced as a sequence of little XML documents info sets. When the XML processing step runs (e.g. XSLT), it caches the streaming of those info sets into a "DOM" tree so that an XML processing step (e.g. XSLT) can run on the whole document. Since that document is tiny, we can process the large data XML document of arbitrary size in constant memory.
6) Service Network Servers (e.g. Ajax server) requests:
    a. Receive XML request with word to complete.

b. Call XML data flow that retrieves list of completion for that word.

c. Format resulting document with XSLT

d. Serialize response to XML.

7) Execute a dynamic query

a. Dynamically create XQuery using XSLT, based on input XML document.

b. Execute XQuery on XML database.

c. Construct XHTML result page based on returned data.

d. Serialize result to HTML

8) Aggregate XML documents

a. Provide several XML documents to aggregate.

b. Perform the aggregation under a new root element.

c. Serialize result as XML.

9) Conditional database access

a. Receive XML document to save on input.

b. Retrieve existing document from the database.

c. If document exists, call the "XUpdate" service.

d. If document does not exist, call the "XInsert" service.

10) Content-dependent transformations

a. Receive document to transform.

b. If document is XHTML, apply the XSLT stylesheet and serialize to HTML.

c. If document is XSL-FO, run XSL-FO to PDF converter.

d. Otherwise, just serialize as XML.

11) Configuration-dependent transformations
    a. Receive document to format on first input.
    b. Receive configuration on second input.
    c. If configuration is “desktop browser”, run the first XSLT transformation and serialize to HTML.
    d. If configuration is “mobile browser”, run the second XSLT transformation and serialize as HTML.
12) Generation of multiple-file command-line document
    a. Read list of source documents.
    b. For each document in list
        i. Read source document.
        ii. Perform series of XSLT transformations.
        iii. Serialize each transformed document to disk, or
        iv. Aggregate all resulting documents and serialize a single resulting document.
13) Importing to a database
    a. Read list of source documents
    b. For each document in list
        i. Validate document.
        ii. Call the “XInsert” service to perform the insertion into a relational or XML database.
14) Supporting a Document Production Framework (DPF) processing sequence
    a. A special purpose tagging task is applied to generate a well formed XML

document (i.e. sectioning).

b. Table of content is extracted from the document; anchors and links are created to specify the document navigation.

c. Pagination is performed, splitting the document structure through some configuration (e.g., 1 sub-section per page, 3 paragraphs per page, etc.), updating the table of content at each pagination step.

d. Each page is transformed into some output language (e.g. XHTML), with the table of content re-integrated.

Figure 2 shows an example of XIE usage that captures a possible XML data flow, where six processing steps are involved. In step 1, an XSLT service processes the incoming XML data with Style Sheet1. In step 2, a similar XSLT service processes the styled data from step 1 with Style Sheet2. In step 3, a broadcast service sends a copy of XML-S2 that is the output from step 2 to the update, route, and some other service. In step 4, the update service executes against a database with an input XML-S2. In step 5, the route service routes the XML-S2 to a browser, and finally in step 6, more processing can be done on the XML-S2 data by a service.

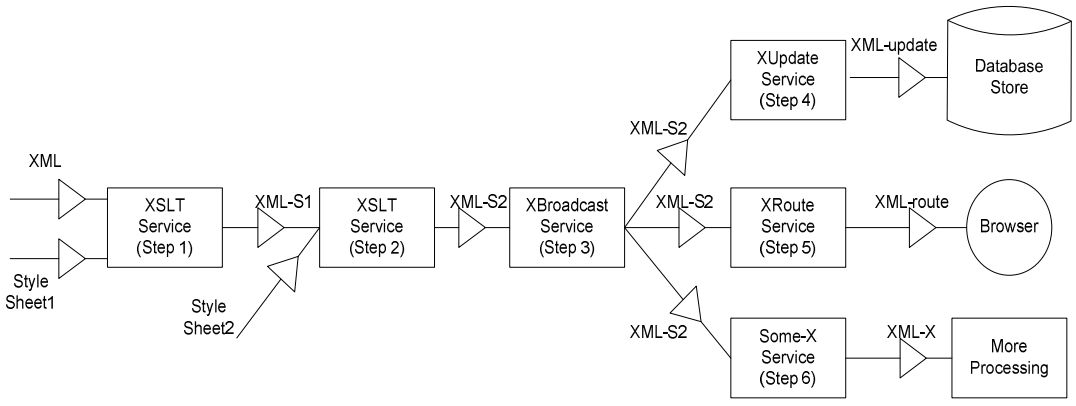


Figure 2: XIE Usage Example

1.4 Contributions

In this research, we proposed XML Integrated Environment (XIE) which is a general-purpose service-oriented architecture for processing schemaless XML documents in a scalable and efficient fashion. The XIE supports a new software service model that provides a proper abstraction to describe a service and divide it into four components: structure, connection, interface and logic. We also proposed and implemented XIE Service Language (XIESL) that can capture the creation and maintenance of the XML processes and the data flow specified by the user and then orchestrates the interactions between different XIE services. XIESL manages the complexity of XML processing by implementing an XML processing pipeline that enables better management, control, interpretation and presentation of the XML data even for non-professional users. The main contributions of this research are as follows:

- Designing and building a general-purpose XML system that can process schemaless XML data that is envisioned to revolutionize the way non-professional programmers see, work and manage their XML assets.
- Providing the foundation for a powerful user-driven data management experience by offering powerful tools and constructs that can fully utilize the XML processing power embedded in the XIE's unified framework and service-oriented architecture.
- Specifying, developing and implementing a unified service-oriented layered architecture for processing XML documents in a scalable, efficient and powerful fashion.

- Designing and implementing a new service model called Meta-Service Model (MSM) that builds the proper abstractions to describe a service that can be easily mapped to the different tiers of the XIE architecture. MSM went one step further than the typical service-oriented models and subdivided the service component into four additional subcomponents: structure, connection, interface and logic.
- Developing and implementing a declarative language called XML Integrated Environment Services Language (XIESL) that describes and specifies the processing of schemaless XML data. XIESL manages the complexity of XML processing by implementing an XML processing pipeline that enables better management, control, interpretation and presentation of the XML data.

1.5 Outline of Dissertation

The rest of the dissertation is structured as follows: Chapter 2 provides detailed background for Service-Oriented Computing (SOC). SOC is the new computing paradigm for distributed computing that utilizes services as fundamental elements for developing applications and solutions. We discuss Service-Oriented Architecture (SOA) that is the new style of architecture used to design enterprise systems. We then explain Web services which are an instantiation of an SOA that has gained widespread industry acceptance. Chapter 3 presents the general-purpose service-oriented layered architecture that is used to build the XML Integrated environment. Chapter 4 introduces the new XML Integrated Environment Service Language (XIESL) with a thorough explanation of its features, benefits, detailed vocabulary and service processing contexts. Chapter 5 illustrates in details the incorporation and integration of the XUpdate service as one of the core XIESL registered services. The XUpdate service implements the XUpdate

specification that defines the syntax and semantics of the XUpdate update language. Chapter 6 presents a few XIE examples. Chapter 7 presents conclusions and summarizes the completed work of XIE. It also describes the future works: Development of a graphical XML data flow designer; incorporating and building more core software service capabilities (e.g. supports for XML schema, compression and encryption); incorporating and building more XIESL core capabilities (e.g. web-service integration); and incorporating a more powerful internal caching mechanism for efficient use and greater flexibility.

Chapter 2
Background

2.1 Service-Oriented Computing (SOC)

Service-Oriented Computing (SOC) is the new computing paradigm for distributed computing that utilizes services as fundamental elements for developing applications and solutions. Services are autonomous platform-independent computational elements that support rapid, low-cost composition of distributed applications. Services can be described, discovered, orchestrated and programmed using XML artifacts for the purpose of developing massively distributed interoperable applications [12].

Services are offered by service providers – organizations that define, develop, deploy, manage and run the service. In general, service providers offer a distributed computing infrastructure for both intra- and cross-enterprise application integration and collaboration.

On the other hand, services are requested and consumed by clients. Clients can be simple or sophisticated processes and applications within an enterprise or outside the enterprise. Consequently, to satisfy the above requirements, services should be:

- *Reusable*: As services are published in a directory available over a network, they become more easily discoverable, reusable and location transparent.
- *Technology neutral*: By using standard based technologies, services will use common denominator technologies for their invocation like standard protocols, descriptions and discovery mechanisms.
- *Loosely-coupled*: Service providers and consumers can be developed independently using well-defined interfaces. Service implementers can change interface, data or message versions in the service without impacting consumers.
- *Non-intrusive in their development*: Existing software components don't need to be modified to expose their functionality as services. Services are developed or generated using the interface definition of a component.
- *Able to offer composite services*: Composite services combine new and existing application logic and transactions [12, 14, 18, 20].

2.2 Service-Oriented Architecture (SOA)

Service-Oriented Architecture (SOA) is a new style of architecture used to design enterprise systems. In the SOA approach, application components are viewed as service consumers or service providers. Services represent complete business functions. They are intended to be used at the level of individual program to the level of the enterprise or even across enterprises. For example, a bank might have a deposit service, a withdrawal service, and a loan approval service. Each service is implemented using one or more application components. In this scenario, a bank customer might interact with a web browser or a customized application that takes on the role of a service consumer. Based

on the action taken by the customer, the service consumer interacts with one or more service providers to complete the customer's transaction.

A service has two parts: the interface and the implementation. The interface defines the programmatic contract between the consumer and the provider. A service interface must contain the identity of the service that will be performed, the format of the input message, and the format of the response message. Thus, service consumer only knows what will be done – not how it will be done. The service implementation contains the functional or business logic of the service. The implementation should be a "black box" to the service consumer.

The SOA defines a well established interaction between service requestors (clients) and service providers. Clients are software agents that request the execution of a service. Providers are software agents that provide the service. Agents can be simultaneously both service clients and providers [12, 24].

The SOA defines a relationship between three kinds of participants: the service provider, the service discovery agency, and the service requestor (client). The interactions involve the publish, find/discover and bind/invoke operations, as shown in Figure 3. These roles and operations act upon the service artifacts: the service description and the service implementation.

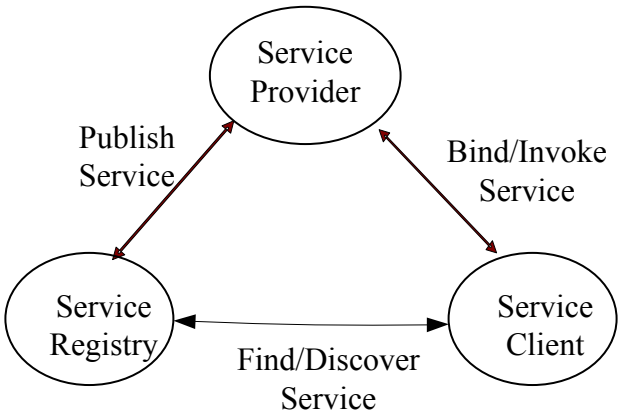


Figure 3: The Basic Service-Oriented Architecture

Service descriptions are used to advertise the service capabilities, interface, behavior, and quality. Publication of such information about available services (on a service registry) provides the necessary means for discovery, selection, binding, and composition of services. In particular, the service interface description publishes the service signature [14, 24, 26].

2.3 SOA and Web Services

The Web Service is an instantiation of an SOA that has gained widespread industry acceptance. However, SOA does not require Web Services for implementation, and Web Services deployment does not result in an SOA. A Web Service is a specific kind of service that is identified by a URI and exhibits the following characteristics:

- It exposes the application's internal workflows, business processes, and architectures over the Internet.
- It can be executed as standalone service, or combined with other services to create composite services or applications [21].

- It uses standard technologies (e.g., standard Internet languages and protocols which are available every where today) and exposes its features programmatically over the Internet.
- It can be implemented via a self-describing interface based on open Internet standards. Interactions of Web Services occur as Simple Object Access Protocol (SOAP) calls over HTTP carrying XML data content, and the service descriptions of the web-services are expressed using Web Service Description Language (WSDL) as the common (XML-based) standard. WSDL is used to publish a Web Service in terms of its operational service descriptions or interfaces. The ports specify the addresses implementing the service. The port types associate an identity for the definition of operation and exchanges of messages. The binding identifies the concrete definition of which packaging and transportation protocols, such as SOAP, are used to interconnect two conversing end-points. In addition, Web Services uses the Universal Description, Discovery and Integration (UDDI) standard. UDDI is a directory service and a network-based repository for service publications and enables Web Service clients to locate candidate services and discover their detailed descriptions as XML interfaces [26].

Chapter 3

XIE Architecture

3.1 The XIE n-tier Architecture

The general-purpose service-oriented layered architecture that is used to build the XML Integrated environment is presented in Figure 4. The XIE is designed and built based on the concept of a software service. Software services are a collection of reusable networked services communicating through well-defined, platform-independent interfaces. XML is used for defining, describing, orchestrating, and capturing the interaction between the different layers. Also, XML is used for application data exchange. The XIE architecture is divided into six different pieces, layers or parts:

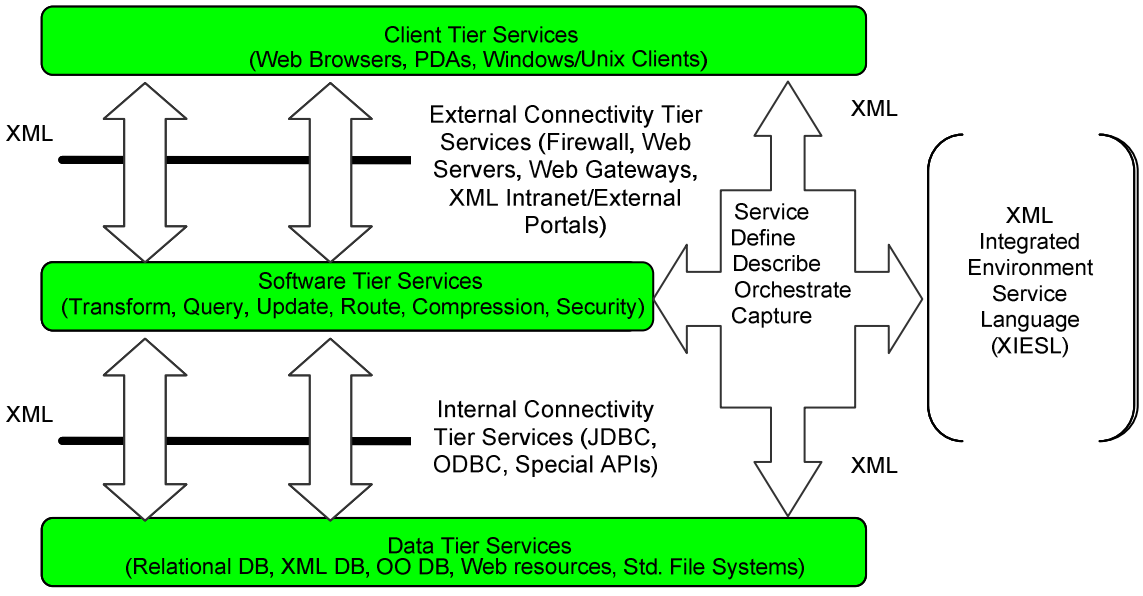


Figure 4: The XML Integrated Environment n-tier Service Layered Architecture

Client Tier abstracts the client or user interfaces and acts as a uniform and ubiquitous information distributor for a wide range of computing devices (such as handheld computers, Personal Digital Assistant (PDAs), cellular telephones, or appliances) and software platforms.

External Connectivity Tier abstracts the external communication between the client tier and the software tier. It provides external connectivity services by allowing us to plug-in different software services when needed. A client or a consumer can bypass this layer and bind to the software tier services directly.

Software Tier abstracts all the software services available directly or over the network. It provides all the core services (e.g. update, query, transform, route, security and compress services) for the XIE from a user's and/or a programmer's perspective. All of these services are described, coordinated and processed through the XML integrated Environment Service Language (XIESL).

Internal Connectivity Tier abstracts the internal communication between the software tier and the data tier. It provides internal connectivity services by allowing us to use standard or non-standard database access methods to retrieve the data requested by a consumer.

Data Tier abstracts the data access services that allow the users to access, integrate, translate, and transform data from various relational and non-relational data resources across the enterprise. These services hide the direct access to the resource, the complexity of the underlying format, and the direct transformation and manipulation of data. They provide a uniform API, a common data-model, and the reuse of consistent information across applications.

XML Integrated Environment Service Language (XIESL) is a declarative language intended to describe and specify the processing of schemaless XML data. It orchestrates the relationships and interactions between the different tasks in the XML data flow to create a business logic capturing the requirements of its users. These tasks are implemented as an XML processing services. An XML processing service is a software component which consumes and produces XML documents. XIESL manages the complexity of XML processing by implementing an XML processing pipeline that enables better management, control, interpretation and presentation of XML data. XML documents are efficiently processed by one or more XML processing services in the XML pipeline, and are then transferred, displayed, translated, parsed, stored, searched or indexed as specified by the XIESL instructions or directives. XIESL creates a unified framework for users to fully utilize the XML processing power embedded in their XML integrated environment.

3.2 XIE Features and Benefits

The XIE has many features and benefits that make the XML processing tasks very easy. The infrastructure is built on top of an optimized XML pipeline engine that uses caching of inputs, outputs and intermediate results for efficient processing of XML documents. This also allows better management and scalability for a large quantity of data. The XIE software services are portable because of their Java implementation and their use of standard XML, XSLT, XUpdate, and XPath parsers and processors. In addition, XIE is designed to support a rich set of XML data composite operations used in XML data flow composition. These operations are implemented efficiently, as part of the XML Integrated Environment Language (XIESL), by a multithreaded data flow engine

with software integrated components and service-oriented architecture. XIESL is simple to use and easy to analyze declarative language that allows better management, control and interpretation of the data.

3.3 The Meta-Service Model

XIE supports a new software service model called the Meta-Service Model (MSM). MSM builds a proper abstraction to describe a service that can be mapped to the different tiers of the XIE architecture. Like the general service-oriented model, MSM divides the services into three major components: consumer, provider and broker as shown in Figure 5. The consumer component binds to the service and calls the provider. The provider offers a rich set of data composite operations or services. The broker publishes the whereabouts of services, and locates services when requested by a consumer. Also, the broker provides all the orchestration, composition and information infrastructure services needed.

MSM goes one step further than typical service-oriented models and subdivides the service component construct into four additional subcomponents: structure, connection, interface, and logic. The structure describes different pieces or parts of the service construct. The connection describes the connection to the service. The interface provides the different methods, operations or tasks provided by the service. The logic represents the actual business logic of the service, including runtime interaction or behavior desired, such as events, method calls, and operations.

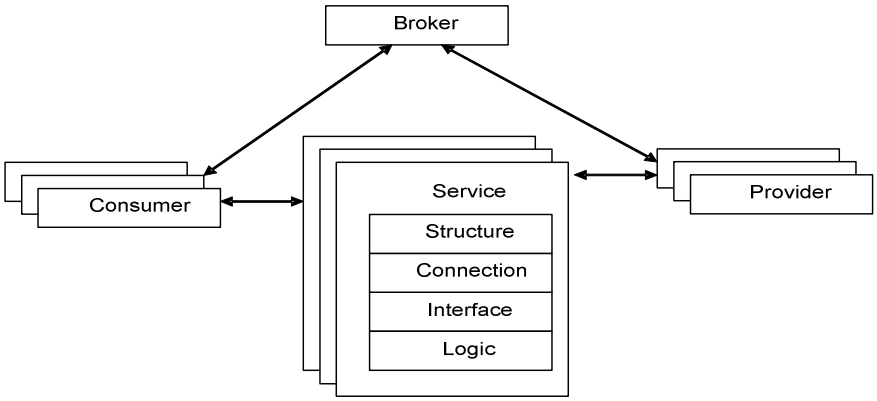


Figure 5: The Meta-Service Model

3.4 The XIE Service-Oriented Architecture

The XIE Service-Oriented Architecture is depicted in Figure 6. The Software Tier Service plays the role of a service provider providing a rich set of XML data composite operations or services. The Client Tier Service takes the role of a client who binds to the service and calls the provider directly. XIESL plays the role of a broker who publishes the whereabouts of services, and locates services when requested by a client. Also, the XIESL provides all the orchestration, composition and information infrastructure services needed for the XML Integrated Environment.

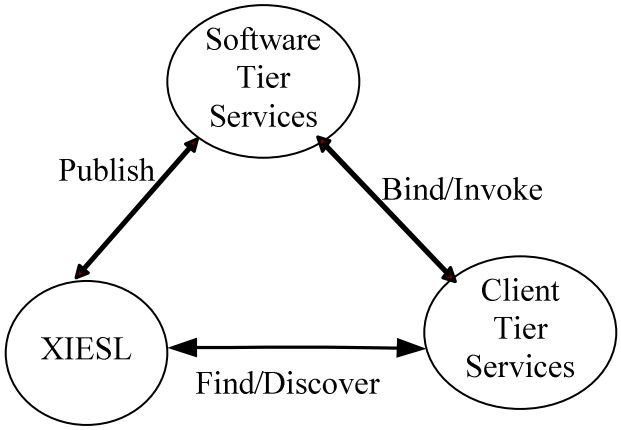


Figure 6: The XIE Service-Oriented Architecture

Chapter 4

XIE Service Language (XIESL)

4.1 XIESL Features and Benefits

XIESL is expressed in XML, so it is possible to author and manipulate XIESL documents using standard XML tools. Also, we are able to address the tasks of capturing, processing and presenting schemaless XML data flows. In addition, XIESL allows better management and control because it is easy and simple to use and to construct new XML documents, extract from existing XML documents, and augment or transform existing XML documents to new XML documents [3]. Moreover, the order of processing tasks can be specified with multiple inputs, multiple outputs and other parameters for the XML processing services that work on the XML documents. XIESL declarative nature allows the building of simple and complex XML data flows using simple statements and directives, which results in increased productivity for complex XML processing.

4.2 XIESL Vocabulary

Figure 7 shows the vocabulary of XIESL, and the details of the vocabulary elements are described in the next section.

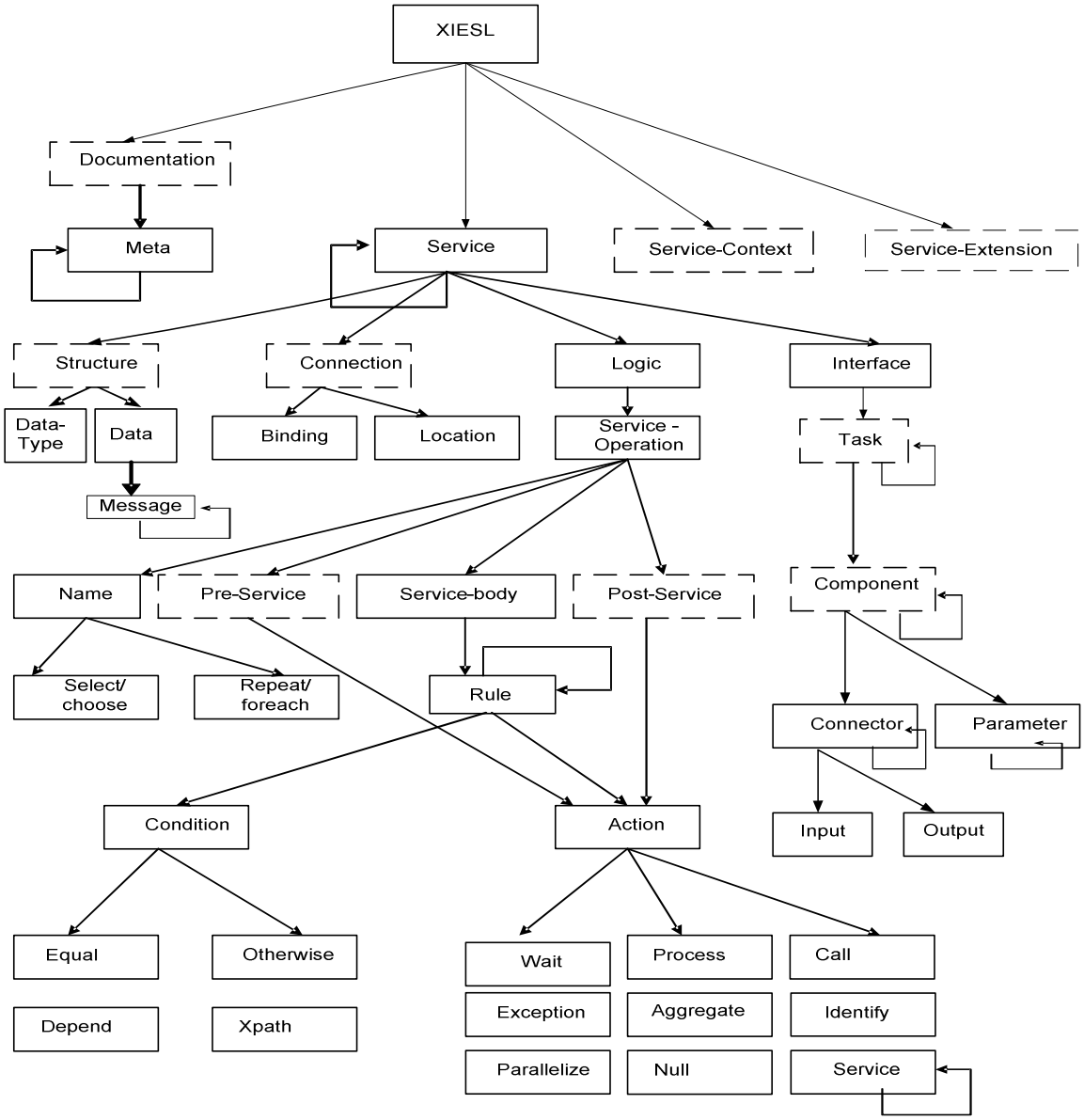


Figure 7: XIESL Vocabulary

4.3 XIESL Details

XIESL introduces the new concept of the data flow processing of XML documents as separate processing software services. XIESL supports the creation and processing of data flow processing services and it handles the structural and processing description.

4.3.1 XIESL pipeline service

The XIESL pipeline service reads the data flow definition conforming to the XIESL syntax. It assembles a pipeline and makes it ready for execution.

4.3.2 XIESL namespace

The XIESL language uses the name space xmlns:x="http://www.wright.edu/xie/pipeline" for all its elements. All the XIESL elements should use the x prefix for consistency and ease of reference.

4.3.3 <x:XIESL> element

The root element of a XIESL document (layout), and it defines:

- General information about the data flow processing services, including human-readable documentation with <x:documentation> .
- Context used for processing documents in the data flow processing services with <x:service-context>.
- The list of services that need to be executed in the pipeline with <x:service>. A service is subdivided into four additional subcomponents: <x:structure>, <x:connection>, <x:interface> and <x:logic>. A service defines a unit of work or a task with its connections to other services in the pipeline using <x:connector>, a condition service using <x:select>, or a repeated service using <x:repeat>.
- Support for particular processing in any given environment without disrupting the current way applications are processing the XIESL documents with the use of the optional <x:service-extension> element, for example, to conform to a particular vocabulary within a specific environment, to support scripting, and to provide

inheritance mechanism to permit XIESL to add (or subtract from) other XIESLs documents.

4.3.4 <x:service> element

The <x:service> element of a XIESL document specifies:

- A unit of work that need to be executed for the pipeline. The service is subdivided into four subcomponents: <x:structure>, <x:connection>, <x:logic> and <x:interface>, that capture the fine-grain details of a service. The <x:structure> describes different pieces or parts of the service construct. <x:connection> describes the connection to the service. <x:logic> represents the actual business logic of the service including runtime interaction, desired service calls or service operations. <x:interface> provides the different methods, operations or tasks provided by the service.
- A name which identifies this service. Different processing services can use this attribute to uniquely identify this service of the XIESL document.

4.3.5 <x:structure> element

The optional <x:structure> element of a XIESL document specifies:

- A container for data type definition using the <x:data-type> element. <x:data-type> is used to specify some type-system used to describe the data used for this service (e.g., schema definition)
- The actual data that is processed and exchanged for this service using the <x:data> element. <x:data> consists of messages that are transmitted for this

service. A message consists of logical parts which is associated with the data type definition container <x:data-type>.

4.3.6 <x:connection> element

The optional <x:connection> element of a XIESL document specifies:

- A concrete protocol and data format specifications for the operations or tasks using the <x:binding> element (e.g., SOAP protocol)
- An address of the service using the <x:location> element. <x:location> specifics the address information of the service (e.g., an URL address)

4.3.7 <x:logic> element

The <x:logic> element of a XIESL document defines:

- The actual business logic of the service including runtime interaction using the <x:service-operation>.

4.3.8 <x:interface> element

The <x:interface> element of a XIESL document specifies:

- The different tasks or operations that is provided by the service using the optional <x:task> element. <x:task> consists of one or more optional components using the <x:component> element.
- A name which identifies this task or operation. Different processing services can use this attribute to uniquely identify this task of the XIESL document.

4.3.9 <x:service-operation> element

The <x:service-operation> element of a XIESL document specifies:

- A service operation that need to be executed for this service. The service operation is subdivided into three subcomponents <x:pre-service>, <x:service-body> and <x:post-service>. <x:pre-service> specifies the pre-processing that is required before the execution of the body of a service. <x:service-body> lists one or more <x:rule> elements that capture the logic of this service. <x:rule> is a container of a rule-based implementation of the logic of the service. <x:rule> specifies the <x:condition> element that should be satisfied for the <x:action> element to execute. <x:post-service> specifies any processing required after the execution of the body of the service.
- A name which identifies this service operation. Different processing services can use this attribute to uniquely identify this service operation of the XIESL document.

4.3.10 <x:rule> element

The <x:rule> element of a XIESL document specifies:

- A container for the processing condition using the <x:condition> element. <x:condition> is used to specify a condition that should be satisfied for the processing to continue.
- The actual action that needs to be performed for this rule using the <x:action> element. <x:action> directs the XIESL engine to perform one of the core operations that is supported by the XIESL language.

4.3.11 <x:condition> element

The <x:condtion> element of a XIESL document specifies:

- An *equal* condition using the <x:equal> element. <x:equal> is used to specify an equal condition that should be satisfied for the processing to continue.
- An *otherwise* condition using the <x:otherwise> element. <x:otherwise> is used to specify an alternative processing path if there is no matching equality.
- A *depend* condition using the <x:depend> element. <x:depend> is used to create a dependency of execution between this condition and some other condition in the XIESL document.
- An xpath result dependency using the <x:xpath> element. <x:xpath> is used to specify an xpath matching expression that is compiled and executed by an xpath service against the XML documents.

4.3.12 <x:action> element

The <x:action> element of a XIESL document specifies processing directives like:

- A wait action using the <x:wait> element. <x:wait> is used to hold processing and wait for the completion of another service execution thread.
- A process action using the <x:process> element. <x:process> is used to direct the continuation of the service execution thread.
- A call action using the <x:call> element. <x:call> is used to call another service execution thread.
- An exception handling action using the <x:exception> element. <x:exception> is used to specify exception handling for the service execution thread.

- An aggregate action using the <x:aggregate> element. <x:aggregate> allows the aggregation of one or more root elements of different XML sets into a one common root element of an XML set.
- An identify action using the <x:identify> element. <x:identify> is used to specify an id for the current executing element in the XIESL document.
- A parallel action using the <x:parallelize> element. <x:parallelize> is used to hint to the XIESL engine the non-existence of any dependency for this rule.
- A null action using the <x:null> element. <x:null> is used to specify that no processing is desired.
- A service action using the <x:service> element. <x:service> is used to specify the unit of work that need to be executed for this pipeline

4.3.13 <x:parameter> element

The <x:parameter> element defines a value pair name and value of a specific variable. It can be used in any processing context within the XIESL document. Each parameter/variable has a name and a value.

- name – The name of the variable.
- value – The value of the variable.

The two variables of an "add" processing component is declared in the XIESL document below:

```
<x:xiesl    xmlns:x="http://www.wright.edu/xie/pipeline">
<x:service  name=”add-at-head-and-at-tail”>
      <x:interface       name=”some-interface”>
            <x:task      name=”some-task”>
                  <x:component    name=”add-component”>
                  <x:parameter    name="position1"          value="head" />
                  <x:parameter    name="position2"          value="tail" />
```

```
                </x:component>
            </x:task>
        </x:interface>
</x:service>
</x:xiesl>
```

The <x:parameter> element and its content are defined in the Relax NG schema with:

```
<define name="parameter">
<zeroOrMore>
        <element name="x:parameter">
                <interleave>
                        <attribute name="name"/>
                        <attribute name="value"/>
                </interleave>
        </element>
</zeroOrMore>
</define>
```

4.3.14 <x:input> element

The <x:input> element specifies and connects the input of a service with the name specified with the name attribute. It enables flexible interactions among services. Each <x:input> element has a name, a role , and an optional href attribute.

- name – Input name which defines an id that can be later referenced.
- role – Specifies the role that this <x:input> plays in the current processing context.

    “data” – a well formed XML data feed from any source (inline, file system, web, etc).

    “current” – a reference to the current data in the processing context.

    “style” – a style sheet.

    “schema” – a schema reference to validate the XML data feed.

    “config” – a configuration setup for the input connector.

"pipe-in" – a pipeline input.

- href – A hyperlink reference to the XML document according to the full syntax of the href attribute.

The <x:input> element and its content are defined in the Relax NG schema with:

```
<define name="input">
    <element name="x:input">
        <attribute name="name"/>
        <attribute name="role"/>
        <optional>
        <attribute name="href"/>
        </optional>
    </element>
</define>
```

4.3.15 <x:output> element

The <x:output> element specifies and connects the output of the service with the name specified with the name attribute. It enables flexible interactions among services. Each <x:output> element has a name, a role, and an optional reference attributes.

- name – Output name which defines an id that can be later referenced.
- role – Specifies the role that this <x:output> plays in the current processing context as follows:

  "id" – output connector identifier.

  "current" – a reference to the current data in the processing context.

  "pipe-connect" – connects the current output to a pipeline output.

  "pipe-out" – a pipeline output.

- reference – Specifies either an id of the current output or a name of the pipeline output that gets connected to this output.

The <x:output> element and its content are defined in the Relax NG schema with:

```
<define name="output">
      <element name="x:output">
            <attribute name="name"/>
            <attribute name="role"/>
            <optional>
            <attribute name="reference"/>
            </optional>
      </element>
 </define>
```

4.3.16 Pipeline inputs and outputs

The pipeline inputs and outputs can be specified by using the <x:connector> and its children: <x:input> and <x:output>. There cannot be two inputs with the same name or two outputs with the same name, but it is possible to have an output and an input with the same name.

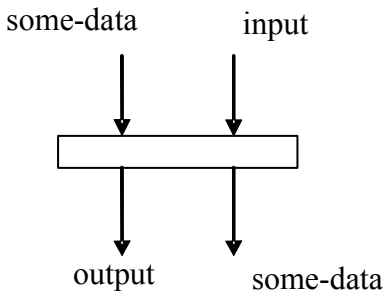


Figure 8: A Pipeline Example

The inputs and outputs of the above pipeline are declared in the XIESL document below:

```
<x:xiesl      xmlns:x="http://www.wright.edu/xie/pipeline”>
<x:service  name=”pipeline-inputs-outputs-example”>
<x:interface        name=”some-interface”>
      <x:task       name=”some-task”>
            <x:component      name=”add-component”>
                  <x:connector       name=”pipeline-connector”>
                        <x:input      name=”some-data”      role="pipe-in"      />
```

```
                    <x:input      name=”input”          role="pipe-in”  />
                    <x:output     name=”some-data”      role="pipe-out"    />
                    <x:output     name=”output”         role="pipe-out"    />
            < /x:connector>
        </x:component>
        </x:task>
</x:interface>
</x:service>
</x:xiesl>
```

4.3.17 <x:component> element

The <x:component> element forms one unit of computation for a service in the pipeline, places a component in the pipeline, and connects it to other components, pipeline inputs, or pipeline outputs.

The following example feeds an XSLT processor with a XML document and an external style sheet. Also, the XML document is validated with a schema while entering the pipeline.

```
<x:xiesl      xmlns:x="http://www.wright.edu/xie/pipeline">
<x:service  name=”style-xml-and-output”>
<x:interface        name=”some-interface”>
<x:task       name=”some-task”>
<x:component     name=”style-an-xml-document”>
<x:connector       name=”pipeline-connector”>
        <x:input     name=”data”       role="current"     href=”xml-input.xml”/>
        <x:input     name=”options”   role="schema”    href="schema-in.xml” />
        <x:input      name=”options” role="style”       href="styelsheet.xml” />
        <x:output     name=”data”      role="pipe-out"    />
        <x:output     name=”result”    role="pipe-out"    reference-with="result-pipe-out"/>
< /x:connector>
</x:component>
</x:task>
</x:interface>
</x:service>
</x:xiesl>
```

The <x:component> element and its content are defined in the Relax NG schema as:

```
<define name="Component">
<element name="x:component">
        <zeroOrOne>
                <ref   name=”x:connector”/>
        </zeroOrOne>
        <interleave>
                <zeroOrMore>
                        <ref   name=”x:input”/>
                </zeroOrMore>
                <zeroOrMore>
                        <ref   name=”x:output”/>
                </zeroOrMore>
                <zeroOrMore>
                        <ref   name=”x:parameter”/>
                </zeroOrMore>
        </interleave>
</element>
</define>
```

4.3.18 <x:select> element

The <x:select> element chooses different software services to execute depending on a specific condition. The general syntax for this is:

```
<x:select   href=”xml-conditioned-doc” xmlns:x="http://www.wright.edu/xie/pipeline">
        <x:when   test="first-condition"   then>...</x:when>
        <x:when   test="second-condition"   then>...</x:when>
        <x:when   test=”third-condition”     then>…</x:when>
          <x:otherwise>...</x:otherwise>
</x:select>
```

The conditions are expressed in XPath and operate on the XML document specified by the href attribute on <x:select>. Each branch can contain one component declaration as well as nested conditions.

The <x:select> element and its content are defined in the Relax NG schema as:

```
<define name="Select">
<element name="x:select">
        <attribute name=”href”/>
        <oneOrMore>
                <element name="x:when">
                <attribute name="test"/>
```

```
            <ref name="Service-to-Execute"/>
            </element>
        </oneOrMore>
        <optional>
        <element name="x:otherwise">
        <ref name="Service-to-Execute"/>
        </element>
        </optional>
</element>
</define>
```

4.3.19 <x:repeat> element

The <x:repeat> element allows us to execute service component multiple times based on the content of a document according to the syntax of the href attribute, selects the desired content according to the Xpath syntax, aggregates results in a root-element with aggregate-with and identifies the current output with the identify-with attribute. The general syntax is:

```
<x:repeat   work-with=”href”                select-with=”Xpath-expression”
            aggregate-with=”root-element”   identify-with=”output-id-name”>
…….
</x:repeat>
```

The <x:repeat> element and its content are defined in the Relax NG schema as:

```
<define name="repeat">
<element name="x:repeat">
        <attribute name=”work-with”/>
        <attribute name=”select-with”/>
        <attribute name=”aggregate-with”/>
        <attribute name=”identify-with”/>
        <oneOrMore>
            <ref name="task">
        </oneOrMore>
</element>
</define>
```

4.3.20 <x:href> attribute

The href attribute is used to:

- Refer to outputs of other processing services.

- Reference external documents.

The complete syntax of the href attribute is described below in a Backus Nauer Form (BNF)-like syntax:

```
href                ::= ( local_reference | external_reference)
local_reference     ::= "#" some-id
external_reference  ::= URI
```

The URI syntax is defined in RFC 2396. A URI is used to reference an external document. A URI can be:

- Absolute, if a protocol is specified. For example, file:/dir/file.xml.

- Relative, if no protocol is specified. For instance ../file.xml. The document is loaded relatively to the URL of the XIESL document where the href is declared.

4.4 XIESL Service Processing Contexts

4.4.1 XIESL integration context

Objectives

- Integrates the data from heterogeneous source (static or dynamic).
- Improves the integrity (physical or data information integration) and semantic integration of XML resources sets.
- Ties things together for handling schemaless structured, unstructured, and semi-structured data.
- Serves as a dynamic integration and incremental server for a user.

Operations

- Scanning – scan for the existence of an element.
- Joining – aggregate two or more XML inputs. The XML inputs are aggregated at the root element.
- Filtering – filters the XML input based on an element.

The complete syntax of the integration attribute is described below in a BNF-like syntax:

```
integration  ::=  scanning | joining | filtering
scanning ::= scan( href element_name )
joining ::= join( href element_name, join_parameter )
join_parameter ::= href [, join_parameter]
filtering ::= filter( href element_name )
element_name::= 'name'
```

4.4.2 XIESL customization context

Objectives

- Customizes the structure and content of the information. For example, multilingual requirements will necessitate the conversion from one language to another.

Operations

- Transformation – transform the data from one format to another using XSL technology.
- Compression – compress the data for compact transfer and storage.
- Securing – secure/check the XML for transfer/processing.

The complete syntax of the customization attribute is described below in a BNF-like syntax:

```
customization ::= transforming | compressing | securing

transforming ::= transform( href element_name, transform_parameter
)

transform_parameter ::= href [, transform_parameter]

compressing  ::= compress( href element_name )

securing ::= secure( href element_name )

element_name::= 'name'
```

4.4.3 XIESL organization context

Objectives

- The ability to route, to slice, and to dice the XML data. For example, we might need to disseminate the XML messages associated with Web Services or a data document associated with large-scale information dissemination systems.

Operations

- Routing – the ability to route the data to one or more targets.
- Excluding – exclude a specific element from the XML collection.
- Including– include a specific element from the XML collection.

The complete syntax of the Organization attribute is described below in a BNF-like syntax:

```
organization::= routing | excluding | including

routing ::= route( href element_name, route_parameter )
```

```
route_parameter ::= href [, route_parameter]

excluding::=  exclude( href element_name )

including::=  include( href element_name )

element_name::=  'name'
```

4.4.4 XIESL generation context

Objectives

- Generates a dynamic user defined XML document structure for a XML integrated environment.
- Analyzes and mines the data to create structured and useable information.
- The ability to track and profile the user's data process flow allowing the creation and adaptation of process improvement plans and strategies. For example, we can learn and create XML vocabularies and tools to detect and resolve discrepancies among XML tags facilitating information integration.

4.5 XIE Server Design

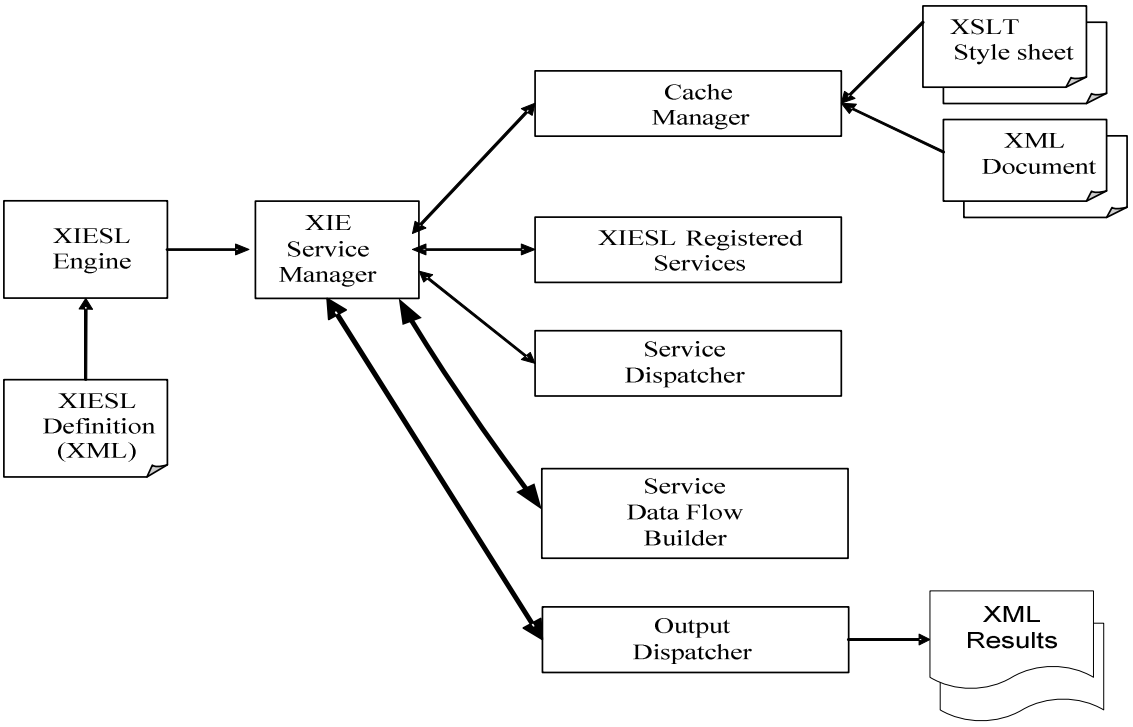


Figure 9: XIE Server Design

Figure 9 shows XIE server design. The XIESL engine is the heart of the XIE system. It reads the data flow definition conforming to the XIESL syntax and then assembles an XML processing pipeline and makes it ready for execution. Also, it implements the Meta-Service Model, manages the complexity of XML processing by implementing an XML processing pipeline, and provides the core XML constructs to enable better management, control, interpretation and presentation of the XML data. The XIESL engine passes control to the XIE service manager that orchestrates and manages the interactions between different XIE services and their supporting infrastructure.

XIE incorporates a cache manager that uses object-level caching associated with XML documents. The cache manager can cache XML transformer objects associated with XSLT transformers in order to save the time-consuming task of compiling a style sheet. Moreover, the cache manager can cache an intermediate document or a resulting XML output from an XML transformer if neither the data source file nor the style sheet has changed. This type of caching is called document-level caching and can usually be implemented at the cost of memory or disk space.

XIESL registered services can customize the structure and content of the information through a transformation service that supports the XSLT 1.0 and 2.0 standards. In addition, there is a select service that can execute different services depending on a condition as specified in the <x:select> element. Moreover, these services have the ability to route, slice, and dice the XML data by Xroute and XPath operations.

Also, XIESL has a registered repeat service that allows the execution of service components multiple times based on the content of a document according to the syntax of the href attribute, selects the desired content according to the Xpath syntax, aggregates results in a root-element with aggregate-with, and identifies the current output with the identify-with attribute. This enhances XIE ability to automate repetitive operations on XML documents and increase productivity and efficiency of complex XML pipelines.

In addition, XIESL registered services incorporate and integrate the XUpdate service. The XUpdate service implements the XUpdate specification that defines the syntax and semantics of the XUpdate update language.

Moreover, XIESL registered services include a join/aggregate operation that supports the aggregation of two or more XML documents. This facilitates easier integration of data from heterogeneous sources and the collection of data results from different services.

Service dispatchers manage and control the actual execution of services including a set of special services called generators with no data inputs but at least one data output and serializers with no data output but at least one data input.

The service data flow builder is responsible for the creation of the XML data flow and considered a helper class for the main XIESL engine. On the other hand, the output dispatcher is a file serializer that is capable of outputting an XML document to a storage resource like a file on a disk.

4.6 XIESL vs. Business Process Execution Language (BPEL)

Business Process Execution Language (BPEL) is an execution language for business processes and a technology that integrates and assembles Web Services [54]. It is an important element of the service-oriented enterprise and the overall Web Service technology stack. BPEL deals explicitly with the functional aspects of business processes: coordinating asynchronous communication between services, correlating message exchanges between parties, implementing parallel processing of activities, manipulating data between partner interactions, supporting long running business

transactions and activities, and providing consistent exception handling [54]. BPEL is defined in an XML format.

XIESL is focused on XML data processing, where the orchestration of relationships and interactions between different tasks in the XML data flow include any XML processing services, including data manipulation, management, control, interpretation and presentation. On the other hand, the usage of BPEL is limited to Web Services as it deals with the functional aspects (e.g. control flow) of business processes that are exposed as Web Services. XIESL provides many built-in XML data integration services (transformation, aggregation and XML updates) that go far beyond the capability of BPEL.

XIESL describes and implements a new software service model called the Meta-Service Model (MSM). XIESL and BPEL share some common ways in describing and composing services like service structure and service interfaces. However, XIESL is unique in the inclusion and supporting of a rule-based processing model under service logic. This allows greater flexibility in the implementation of the actual business logic of the service, including the ability to change the runtime interaction or behavior desired, by simply modifying or inserting new rules.

Both XIESL and BPEL empower the selection of the best-of-breed tools, processes and services to incorporate into the business operation. For XIESL, this provides flexibility to add, replace, or upgrade new and existing XML processing services. For instance, we can replace the current XSLT service with a more efficient one, or add a new XQuery service to the set of built-in services provided by the XML Integrated Environment (XIE). For BPEL, this provides flexibility to replace or upgrade

certain aspects of a business process without impacting the systems that are working well. For instance, a company can change their warehouse service provider without impacting their order management system, even though both may be participants in several business processes [54].

Chapter 5

XUpdate Service

5.1 XIESL XUpdate Service

The XUpdate service implements the XUpdate specification that defines the syntax and semantics of the XUpdate language. This language describes how to update an XML document and makes extensive use of the expression language defined by XPath. An update is represented by an *xupdate:modifications* element in an XML document. *xupdate:modifications* must have a version attribute, indicating the version of XUpdate that the update requires. For this version of XUpdate, the value should be 1.0.

The *xupdate:modifications* element may contain the following types of elements [44]:

- *xupdate:insert-before*
- *xupdate:insert-after*
- *xupdate:append*
- *xupdate:update*
- *xupdate:remove*
- *xupdate:rename*
- *xupdate:variable*
- *xupdate:value-of*
- *xupdate:if*

5.2 Why XUpdate?

In general, XUpdate transformation is similar to XSLT transformation. Both transform an XML input document into an XML output document based on some configuration. With XSLT, the configuration is an XSLT stylesheet. With XUpdate, the configuration is a program written in the XUpdate language. So, when is it appropriate to use XUpdate and when is it appropriate to use XSLT?

XUpdate is more appropriate when the output document is similar to the input document, while XSLT is more appropriate when the output document is a new document in which values from the input document are inserted.

From an execution model perspective, XSLT nodes from the input document trigger the execution of templates that define the output document, while XUpdate provides the sequence of operations to be performed on the input document to create the output document.

From an application development perspective, XSLT can be used as a template language, for instance to create HTML page with both dynamic and static data. The input document contains the dynamic data. The stylesheets contain the static data and describes how the dynamic data is inserted in the document. Doing the same thing in XUpdate would be unnecessarily complicated.

XUpdate can be used as an annotation language, for instance to validate elements of an input document and to add attributes on those elements stating if they are valid or not. Another usage is when annotating an XML document to add calculated values based on existing values. Some cases, for instance when parent elements have to be updated

based on the updates previously done to child elements, cannot be handled in XSLT 1.0 without using XSLT extensions or pipelining multiple stylesheets.

5.3 Using the XUpdate Service

5.3.1 Interface

The XUpdate service interface has two inputs: configuration and XUpdate program, and data: the document to update. In XIESL, the XUpdate service invocation is as follows:

XIESL definition:

```
<x:xiesl  xmlns:x= "http://www.wright.edu/xie/pipeline"
        xmlns:xie="http://www.wright.edu/xie/services">
<x:service name="xie:xupdate">
   <x:interface name="some-interface" >
      <x:input name="options" role="config" href="xupdate.xml"/>
      <x:input name="data" role="data" href="#data"/>
      <x:output name="data" role="pipe-out" id="output"/>
  </x:interface>
</x:service>
</x:xiesl>
```

5.3.2 Using Multiple Documents

The XUpdate service can connect to and process multiple input documents, where a document need to be updated based on information stored in other documents. For example, Figure 10 illustrates the situation where a document is modified based on four other documents: a, b, c, and d.

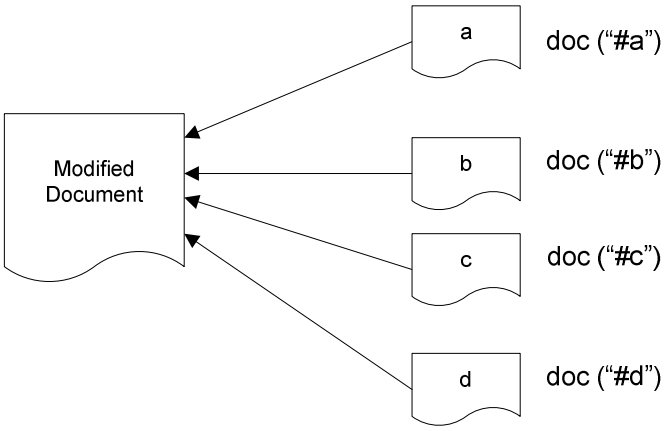


Figure 10: Using Multiple Documents in XUpdate

The Xupdate service program has access to the those documents through XPath expressions with the document() or doc() functions using the URIs: #a, #b, #c, and #d [44, 54]. In XIESL, the XUpdate service invocation is as follows:

XIESL definition:

```
<x:xiesl  xmlns:x= "http://www.wright.edu/xie/pipeline"
        xmlns:xie="http://www.wright.edu/xie/services">
<x:service name="xie:xupdate">
   <x:interface name="some-interface" >
      <x:input name="options" role="data" href="xupdate.xml"/>
      <x:input name="data" role="data" href="#data"/>
      <x:input name="a" role="data" href="a.xml"/>
      <x:input name="b" role="data" href="b.xml"/>
      <x:input name="c" role="data" href="c.xml"/>
      <x:input name="d" role="data" href="d.xml"/>
      <x:output name="data" role="pipe-out" id="output"/>
  </x:interface>
</x:service>
</x:xiesl>
```

5.4 XUpdate Usage Cases

The set of usage cases for the XUpdate specification given in this section covers the basic functionality and is intended as a solid set of XUpdate examples showing how the language can be used [55].

All usage cases operate on the following XML document that is stored in a file named addresses.xml.

addresses.xml:

```
<addresses>
  <address id="1">
    <!--This is the users name-->
    <name>
      <first>John</first>
      <last>Doe</last>
    </name>
    <city>Dayton</city>
    <state>Ohio</state>
    <country>United States</country>
    <phone type="home">937-433-0300</phone>
    <phone type="work">937-555-6070</phone>
    <note><![CDATA[This is a new user]]></note>
  </address>
  </addresses>
```

All usage cases were tested against XIESL XUpdate service.

5.4.1 Insert Element Before

Add a middle name element before the last name element. The XUpdate definition is stored in a file called XIE-Insert-Before.xml.

XIE-Insert-Before.xml:

```
<xu:modifications   xmlns:xu="http://www.xmldb.org/xupdate">
    <xu:insert-before   select="/addresses/address[@id=1]/name/last">
        <xu:element name="middle">Steven</xu:element>
    </xu:insert-before>
</xu:modifications>
```

Here is the XUpdate service invocation in XIESL:

XIESL definition:

```
<x:xiesl  xmlns:x= "http://www.wright.edu/xie/pipeline"
        xmlns:xie="http://www.wright.edu/xie/services">
<x:service name="xie:xupdate">
   <x:interface name="some-interface" >
      <x:input name="options" role="data" href="XIE-Insert-Before.xml"/>
      <x:input name="data" role="data" href="#update-xml-input"/>
      <x:output name="data" role="pipe-out" id="update-XML-output"/>
  </x:interface>
</x:service>
</x:xiesl>
```

5.4.2 Insert Element After

Add a cell-phone element after the home phone element.

XUpdate definition:

```
<xu:modifications   xmlns:xu="http://www.xmldb.org/xupdate">
  <!-- Add a cell phone element after the home phone number-->
    <xu:insert-after   select="/addresses/address[@id=1]/phone[@type='home']">
        <xu:element name="phone">
          <xu:attribute name="type">cell</xu:attribute>
          937-494-4904
        </xu:element>
    </xu:insert-after>
</xu:modifications>
```

5.4.3 Append Element

Append a zip code element to the address record.

XUpdate definition:

```
<xu:modifications   xmlns:xu="http://www.xmldb.org/xupdate">
  <!-- Append a zip code element to the address record -->
    <xu:append   select="/addresses/address[@id=1]">
        <xu:element name="zip">45449</xu:element>
    </xu:append>
</xu:modifications>
```

5.4.4 Insert attribute

Add an extension attribute to the work phone element.

XUpdate definition:

```
<xu:modifications   xmlns:xu="http://www.xmldb.org/xupdate">
  <!-- Add an extension attribute to the work phone element -->
    <xu:append   select="/addresses/address[@id=1]/phone[@type='work']">
        <xu:attribute name="extension">122</xu:attribute>
    </xu:append>
</xu:modifications>
```

5.4.5 Insert XML Block

Add a new address record to the top-level address element.

XUpdate definition:

```
<xu:modifications   xmlns:xu="http://www.xmldb.org/xupdate">
  <!-- Add a new address record to the top level addresses element -->
    <xu:append   select="/addresses">
      <xu:element name="address">
        <xu:attribute name="id">2</xu:attribute>
        <name>
          <first>Susan</first>
          <last>Wang</last>
        </name>
        <city>Columbus</city>
        <state>Ohio</state>
        <country>United States</country>
        <phone type="home">513-504-2030</phone>
      </xu:element>
    </xu:append>
</xu:modifications>
```

5.4.6 Update Element

Change the first name of address with id = 1 to be Johnathan.

XUpdate definition:

```
<xu:modifications   xmlns:xu="http://www.xmldb.org/xupdate">
  <!-- Change the first name of address with id = 1 to Johnathan -->
    <xu:update  select="/addresses/address[@id=1]/name/first">Johnathan</xu:update>
</xu:modifications>
```

5.4.7 Update Attribute

Change the type of the phone number 937-433-0300 to be a cell.

XUpdate definition:

```
<xu:modifications   xmlns:xu="http://www.xmldb.org/xupdate">
  <!-- Change the type of the phone number 937-433-0300 to be a cell -->
    <xu:update  select="/addresses/address[@id=1]/phone[.='937-433-
0300']/@type">cell</xu:update>
</xu:modifications>
```

5.4.8 Delete Element

Remove all phone elements.

XUpdate definition:

```
<xu:modifications   xmlns:xu="http://www.xmldb.org/xupdate">
  <!-- Remove all phone elements -->
    <xu:remove  select="/addresses/address[@id=1]/phone"/>
</xu:modifications>
```

5.4.9 Delete Attribute

Delete all type attributes on phone elements.

XUpdate definition:

```
<xu:modifications   xmlns:xu="http://www.xmldb.org/xupdate">
  <!-- Remove all type attributes on phone elements -->
    <xu:remove  select="/addresses/address[@id=1]/phone/@type"/>
</xu:modifications>
```

5.4.10 Copying a Node

Copy the state node and place the copy after the country node.

XUpdate definition:

```
<xu:modifications   xmlns:xu="http://www.xmldb.org/xupdate">
  <!-- Copy the state node and place the copy after the country node -->
    <xu:variable   name="state"  select="/addresses/address[@id=1]/state" />
    <xu:insert-after   select="/addresses/address[@id=1]/country">
      <xu:value-of  select="$state"/>
    </xu:insert-after>
</xu:modifications>
```

5.4.11 Moving a Node

Move the country node before the state node.

XUpdate definition:

```
<xu:modifications   xmlns:xu="http://www.xmldb.org/xupdate">
  <!-- Move the country node before the state node -->
    <xu:variable   name="country"  select="/addresses/address[@id=1]/country" />
    <xu:remove   select="/addresses/address[@id=1]/country"/>
    <xu:insert-before  select="addresses/address[@id=1]/state">
      <xu:value-of  select="$country"/>
    </xu:insert-before>
</xu:modifications>
```

Chapter 6

XIE Examples

6.1 A Simple Pipeline Service

Consider the following XML document containing a simple title element:

```
<document>
   <title>This is a simple XIE example</title>
</document>
```

You may want to read this document as an inline XML document and store it on your local file system under the c:\temp directory.

The XIESL for doing that is as follows, and the graphical depiction is shown in Figure 11.

The XIESL definition:

```
<?xml                version="1.0" encoding="ISO-8859-1" ?>
<!--   A Simple XIE example -->
<x:xiesl  xmlns:x="http://www.wright.edu/xie/pipeline"
        xmlns:xie="http://www.wright.edu/xie/services">
<x:service name="xie:file-serializer">
        <x:structure                                                    name="some-structure"/>
```

```
        <x:connection name="some-connection"/>
        <x:logic name="some-logic"/>
        <x:interface name="some-interface" >
          <x:task  name="some-task" >
                <x:component name="some-component">
                <x:input name="options" role="config">
                        <options>
                         <content-type>text/xml</content-type>
                        <directory>c:/temp</directory>
                         <file>XIE-Simple-xiesl.xml</file>
                        </options>
                </x:input>
                <x:input name="data" role="current">
                <document>
                        <title>This is a simple XIE example</title>
                </document>
                </x:input>
                </x:component>
  </x:task>
  </x:interface>
  </x:service>
</x:xiesl>
```

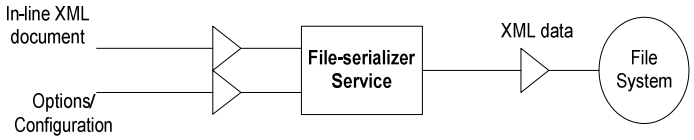


Figure 11: A Simple Pipeline Example

Here are the execution steps for the pipeline:

1. The in-line XML document and the configuration options inputs are read by the file-serializer service.
2. The file-serializer service executes storing the XML data in a file named XIE-Simple-xiesl.xml under the c:/temp directory.

XIE-Simple-xiesl.xml:

```
<?xml version="1.0" encoding="utf-8"?>
        <document                                   xmlns:xie=http://www.wright.edu/xie/services
        xmlns:x="http://www.wright.edu/xie/pipeline">
        <title>This is a simple XIE example</title>
</document>
```

6.2 A Select/Choose Service

Consider an XML document containing a simple execute element with a child element called condition1. This document is stored in a file named choose-input.xml.

Choose-input.xml:

```
<execute>
   <condition1>1</condition1>
</execute>
```

You may want to direct the execution path of the select/choose service depending on the value of the test condition branch in the select statement.

The XIESL for doing that is as follows, and the graphical depiction is shown in Figure 12.

The XIESL definition:

```
<?xml                version="1.0" encoding="ISO-8859-1" ?>
<!-- Select/Choose Example -->
<x:xiesl xmlns:x="http://www.wright.edu/xie/pipeline"
        xmlns:xie="http://www.wright.edu/xie/services">

  <x:select   href="choose-input.xml"  xmlns:x="http://www.wright.edu/xie/pipeline">
    <x:when   test="/execute/condition1">
        <x:service name="xie:file-serializer">
        <x:interface name="some-interface">
        <x:input name="options" role="config">
           <options>
              <content-type>text/xml</content-type>
              <directory>c:/temp</directory>
              <file>XIE-choose-condition1.xml</file>
           </options>
        </x:input>
        <x:input name="data" role="config">
```

```
          <document>
             <title>This is a condition1 example from the choose service</title>
          </document>
        </x:input>
       </x:interface>
     </x:service>
    </x:when>
    <x:when   test="/execute/condition2">
       <x:service name="xie:file-serializer">
        <x:interface name="some-interface">
        <x:input name="options" role="config">
           <options>
              <content-type>text/xml</content-type>
              <directory>c:/temp</directory>
              <file>XIE-choose-condition2.xml</file>
           </options>
        </x:input>
        <x:input name="data" role="current">
         <document>
            <title>This is a condition2 example from the choose service</title>
         </document>
        </x:input>
     </x:interface>
     </x:service>
    </x:when>
    <x:otherwise>
      <x:service name="xie:file-serializer">
        <x:interface name="some-interface">
        <x:input name="options" role="config">
           <options>
              <content-type>text/xml</content-type>
              <directory>c:/temp</directory>
              <file>XIE-choose-otherwise.xml</file>
           </options>
        </x:input>
        <x:input name="data" role="current">
         <document>
            <title>This is otherwise example from the choose service</title>
         </document>
        </x:input>
     </x:interface>
     </x:service>
   </x:otherwise>
 </x:choose>
</x:xiesl>
```

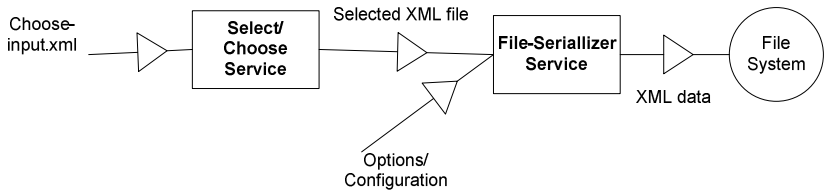


Figure 12: A Select/Choose Service

Here are the execution steps for the pipeline:

1. The choose-input.xml file is read by the select/choose service: the test condition <x:when test="/execute/condition1"> evaluates to true.
2. The corresponding file-serializer service executes storing the XML data in a file named XIE-choose-condition1.xml under the c:/temp directory.

XIE-choose-condition1.xml:

```
<?xml version="1.0" encoding="utf-8"?>
        <document                    xmlns:xie="http://www.wright.edu/xie/services"
        xmlns:x="http://www.wright.edu/xie/pipeline">
        <title>This is a condition1 example from the choose service</title>
</document>
```

6.3 An XML pipeline with two XSLT services

Consider an XML document containing CD catalog information like title, artist, country, company, price and year. This document is stored in a file named cdcatalog.xml.

cdcatalog.xml:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<catalog>
        <cd>
                <title>Empire Burlesque</title>
                <artist>Bob Dylan</artist>
                <country>USA</country>
```

```
        <company>Columbia</company>
        <price>10.90</price>
        <year>1985</year>
    </cd>
    <cd>
        <title>Hide your heart</title>
        <artist>Bonnie Tyler</artist>
        <country>UK</country>
        <company>CBS Records</company>
        <price>9.90</price>
        <year>1988</year>
    </cd>
      .... the rest of the xml data ….
```

You may want to filter the artist and the title elements using a style sheet named cdcatalog.xsl. This document is stored in a file named cdcatalog.xsl.

cdcatalog.xsl:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:template match="/">
<catalog>
    <xsl:for-each select="catalog/cd">
    <cd>
      <title><xsl:value-of select="title"/></title>
      <artist><xsl:value-of select="artist"/></artist>
    </cd>
    </xsl:for-each>
</catalog>
</xsl:template>
</xsl:stylesheet>
```

Then, you may want to use the filtered output and sort it on the artist element using a style sheet named cdcatalaog_sort.xsl. This document is stored in a file named cdcatalaog_sort.xsl.

cdcatalog_sort.xsl:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet version="1.0"  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:template match="/">
<html>
```

```
 <body>
  <h2>My CD Collection</h2>
  <table border="1">
   <tr bgcolor="#9acd32">
    <th>Title</th>
    <th>Artist</th>
   </tr>
   <xsl:for-each select="catalog/cd">
   <xsl:sort select="artist" />
   <tr>
    <td><xsl:value-of select="title"/></td>
    <td><xsl:value-of select="artist"/></td>
   </tr>
   </xsl:for-each>
  </table>
 </body>
 </html>
</xsl:template>
</xsl:stylesheet>
```

The XIESL for doing all of the above is as follows, and the graphical depiction is shown in Figure 13.

The XIESL definition:

```
<?xml                version="1.0" encoding="ISO-8859-1" ?>
<x:xiesl xmlns:x="http://www.wright.edu/xie/pipeline"
        xmlns:xie="http://www.wright.edu/xie/services">
  <!-- Pipeline input called "cdcatalog-input" -->
  <x:parameter  name="cdcatalog-input" role="pipe-in" />
  <x:service name="xie:xslt">
   <x:interface>
    <x:connector>
     <x:input    name="data"       role="current"   href="cdcatalog.xml"/>
     <x:input    name="options"    role="style"     href="cdcatalog.xsl"/>
     <x:output   name="data"       role="current"   with-id="style1-XML-output"/>
    </x:connector>
   </x:interface>
  </x:service>
 <x:service name="xie:xslt">
    <x:interface>
    <x:connector>
     <x:input     name="data"       role="current"    href="#style1-XML-output"/>
     <x:input     name="options"    role="style"      href="cdcatalog_sort.xsl"/>
     <x:output    name="data"       role="current"    with-id="style2-XML-output"/>
```

```
    </x:connector>
     </x:interface>
   </x:service>
</x:xiesl>
```

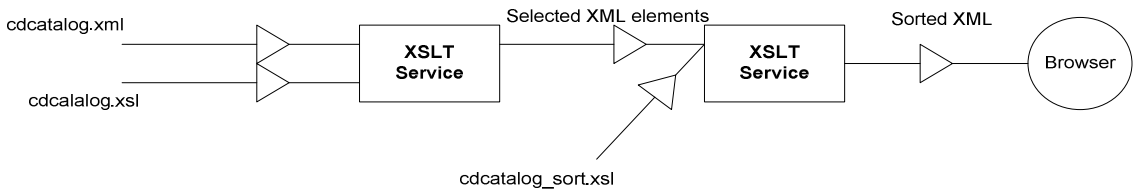


Figure 13: A Pipeline including XSLT(pick) and XSLT(sort) Services

Here are the execution steps for the pipeline:

1. The cdcatalog.xml input document is read by the first XSLT service.
2. The cdcatalog.xsl style sheet is read and executed: the title and artist elements are selected from the cd catalog input.
3. The selected data is passed to the second XSLT service.
4. The cdcatalog_sort.xsl style sheet is read and executed by the second XSLT service: the elements are sorted on the artist element.
5. The sorted cd catalog data is passed along and displayed by the browser as depicted in Figure 14 below.

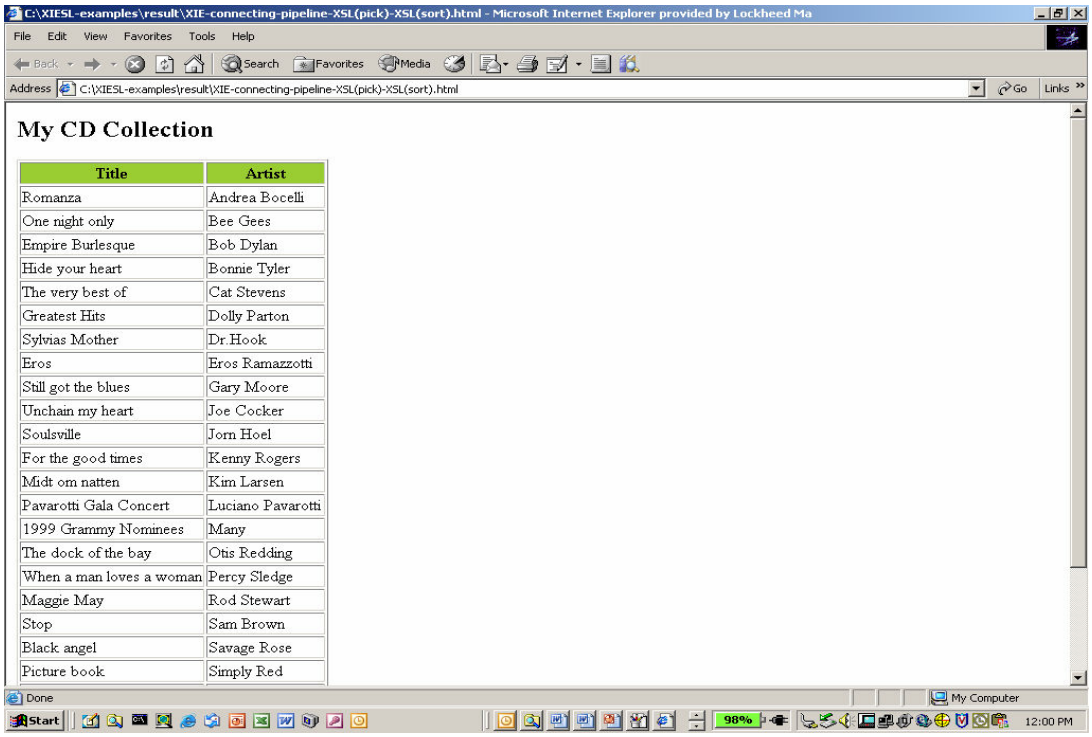


Figure 14: A Sorted CD Catalog

6.4 A Join/Aggregate Service

Consider two or more XML documents that needed to be joined or aggregated at the root level. These documents can be either static or dynamic inputs to the join/aggregate function. Those XML documents are stored in two files named aggregate-document1.xml and aggregate-document2.xml.

Aggregate-document1.xml:

```
<document1>
<title1>This is an aggregate-document 1 for the Join/Aggregate service </title1>
 </document1>
```

Aggregate-document2.xml:

```
<document2>
   <title2>This is an aggregate-document 2 for the Join/Aggregate Service</title2>
```

</document2>

You may want to simply join or aggregate the two documents at the root level. The XIESL for doing that is as follows, and the graphical depiction is shown in Figure 15.

The XIESL definition:

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!-- Join/Aggregate Example -->
<x:xiesl xmlns:x="http://www.wright.edu/xie/pipeline"
        xmlns:xie="http://www.wright.edu/xie/services">
   <x:service name="xie:url-generator" xmlns:x="http://www.wright.edu/xie/pipeline">
   <x:interface name=”url-interface”>
      <x:input name="options" role=”config”>
         <options>
            <url>xie:aggregate-document1.xml</url>
            <content-type>application/xml</content-type>
            <validating>false</validating>
         </options>
      </x:input>
      <x:output name="data1" role=”id” />
  </x:interface>
</x:service>
    <x:service name="xie:url-generator" xmlns:x="http://www.wright.edu/xie/pipeline">
      <x:input name="options" role=”config”>
         <options>
            <url>xie:aggregate-document2.xml</url>
            <content-type>application/xml</content-type>
            <validating>false</validating>
         </options>
      </x:input>
      <x:output name="data2" role="id"/>
   </x:service>
   <x:service name="xie:file-serializer">
      <x:input name="options" role=”config”>
         <options>
            <content-type>text/xml</content-type>
            <directory>c:/temp</directory>
            <file>Simple-Aggregate.xml</file>
         </options>
      </x:input>
      <x:input name="aggregate-data" role=”data”  href="join('aggregate-document',
#data1, #data2)"/>
   </x:service>
```

</x:xiesl>

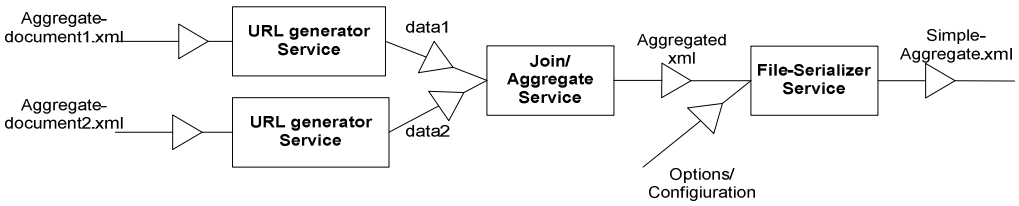


Figure 15: A Joint/Aggregate Service

Here are the execution steps for the pipeline:

1. The aggregate-document1.xml and aggregate-document2.xml files are read by the URL generator services: data1 and data2 are generated respectively as outputs.
2. The join/aggregate service executes with data1 and data2 as XML inputs: the two inputs are joined or aggregated at the root element (aggregate-document), and the data is passed to the file-serializer service.
3. The file-serializer service stores the XML data in a file named Simple-Aggregate.xml under the c:/temp directory.

Simple-Aggregate.xml:

```
<?xml    version="1.0" encoding="ISO-8859-1" ?>
<aggregate-document>
        </document1>
                <title1>This is an aggregate-document 1 for the Join/Aggregate service
          </title1>
        </document1>
          <document2>
                  <title2>This is an aggregate-document 2 for the Join/Aggregate
          Service</title2>
          </document2>
</aggregate-document>
```

6.5 A Quarterly Report Service

Consider an XML document containing information about graduate students in the department of Computer science and Engineering. This document is stored in a file called cse.xml. You may want to print a quarterly report using a style sheet called cse-quarterly.xsl for each student. The XIESL for doing this is as follows:

The XIESL definition:

```
<?xml                version="1.0" encoding="ISO-8859-1" ?>
<x:xiesl      xmlns:x="http://www.wright.edu/xie/pipeline"
                 xmlns:xie="http://www.wright.edu/xie/services">
<x:service  name="monthly-report-service">
<x:structure/>
<x:connection/>
<x:logic>
<x:service-operation name="repeat" >
 <x:pre-service>
   <x:aggregate   aggregate-with="report"/>
   <x:identify     identify-with="report-out"/>
</x:pre-service>
<x:service-body>
  <x:rule>
    <x:condition>
      <x:xpath  work-with="cse.xml"  select-with="/cse/student" />
     </x:condition>
    <x:action>
      <x:process/>
    </x:action>
 </x:rule>
</x:service-body>
<x:post-Service>
 <x:null/>
</x:post-Service>
</x:service-operation>
</x:logic>
<x:interface>
<x:task                name="monthly-report">
<x:component    name ="Transform">
<x:connector>
   <x:input     name="cse-data"          role="input"          href="current()"/>
   <x:input     name="cse-config"        role="style"          href="cse-monthly.xsl"/>
```

```
   <x:output name="cse-data" role="pipe-out" reference-with="report-out" />
</x:connector>
</x:component>
</x:task>
</x:interface>
</x:service>
</x:xiesl>
```

Here is the processing sequence according to the XIESL document shown above:

1. The cse.xml input document is referenced by the attribute *work-with*.
2. The different parts of the documents (different students) are selected based on the *select-with* attribute.
3. The Transform component gets executed on each part (each student element).
4. The results (XML documents) produced by the execution of each component are aggregated in the root element report (*aggregate-with* attribute).
5. The aggregated document report-out is accessible with the *identify-with* attribute.

6.6 An Order Processing Pipeline

Consider an XML document containing orders information for an enterprise like OrderID, CustomerID, EmployeeID, OrderDate, RequiredDate, ShippedDate, ShipVia, Freight, ShipName, ShipAddress, ShipCity, ShipPostalCode, and ShipCountryRegion. The document is generated by joining or aggregating XML order files from different regions of the enterprise. This document is stored in a file named orders.xml.

You may want to filter the OrderID, CustomerID, OrderDate, RequiredDate, ShippedDate, ShipName, ShipCity and the ShipCountryRegion elements using a style sheet named orders.xsl. This document is stored in a file named orders.xsl.

orders.xsl:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:template match="/">
<allOrders>
    <xsl:for-each select="allOrders/orders">
    <orders>
      <OrderID><xsl:value-of select="@OrderID"/></OrderID>
      <CustomerID><xsl:value-of select="@CustomerID"/></CustomerID>
      <OrderDate><xsl:value-of select="@OrderDate"/></OrderDate>
      <RequiredDate><xsl:value-of select="@RequiredDate"/></RequiredDate>
      <ShippedDate><xsl:value-of select="@ShippedDate"/></ShippedDate>
      <ShipName><xsl:value-of select="@ShipName"/></ShipName>
      <ShipCity><xsl:value-of select="@ShipCity"/></ShipCity>
      <ShipCountryRegion><xsl:value-of
select="@ShipCountryRegion"/></ShipCountryRegion>
    </orders>
    </xsl:for-each>
</allOrders>
</xsl:template>
</xsl:stylesheet>
```

Then, you may want to use the filtered output and update the ShipCity element (e.g. update ShipCity element for OrderID 10716 to Dayton) using an XUpdate program named order_update.xml. This document is stored in a file named order_update.xml.

order_update.xml:

```
<xu:modifications   xmlns:xu="http://www.xmldb.org/xupdate">
 <!-- Update the ShipCity for OrderID 10716 to Dayton -->
 <xu:update   select="allOrders/orders[OrderID='10716']/ShipCity">Dayton</xu:update>
</xu:modifications>
```

Then, you may want to use the updated output and sort it on the ShipCountryRegion, ShipCity and RequiredDate elements using a style sheet named orders_sort.xsl. This document is stored in a file named orders_sort.xsl.

orders_sort.xsl:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:template match="/">
```

```
 <html>
 <body>
  <h2>All Orders</h2>
  <table border="1">
   <tr bgcolor="#9acd32">
    <th>OrderID</th>
    <th>CustomerID</th>
    <th>OrderDate</th>
    <th>RequiredDate</th>
    <th>ShippedDate</th>
    <th>ShipName</th>
    <th>ShipCity</th>
    <th>ShipCountryRegion</th>
   </tr>
   <xsl:for-each select="allOrders/orders">
   <xsl:sort select="ShipCountryRegion" />
   <xsl:sort select="ShipCity" />
   <xsl:sort select="RequiredDate" />
   <tr>
    <td><xsl:value-of select="OrderID"/></td>
    <td><xsl:value-of select="CustomerID"/></td>
    <td><xsl:value-of select="OrderDate"/></td>
    <td><xsl:value-of select="RequiredDate"/></td>
    <td><xsl:value-of select="ShippedDate"/></td>
    <td><xsl:value-of select="ShipName"/></td>
    <td><xsl:value-of select="ShipCity"/></td>
    <td><xsl:value-of select="ShipCountryRegion"/></td>
   </tr>
   </xsl:for-each>
  </table>
 </body>
 </html>
</xsl:template>
</xsl:stylesheet>
```

The core XIESL services for doing all of the above is as follows, and the graphical depiction is shown in Figure 16.

The XIESL definition:

```
<x:service name="xie:xslt">
  <x:interface name="some-interface">
    <x:input name="data" role="data" href="aggregate('all', #ordersEast, #ordersWest, #ordersNorth, #ordersSouth)"/>
    <x:input name="options" role="config" href="orders.xsl"/>
```

```
    <x:output name="data" role="data" id="style1-XML-output"/>
 </x:interface>
 </x:service>
<x:service name="xie:xupdate">
 <x:interface name="some-interface" >
    <x:input name="options" role="data" href="XIE-order-update.xml"/>
    <x:input name="data" role="data"  href="#style1-XML-output"/>
    <x:output name="data" role="pipe-out" id="update-XML-output"/>
</x:interface>
</x:service>
<x:service name="xie:xslt">
<x:interface name="some-interface">
    <x:input name="data" role="data" href="#update-XML-output"/>
    <x:input name="options" role="config" href="orders_sort.xsl"/>
    <x:output name="data" role="data" id="style2-XML-output"/>
</x:interface>
 </x:service>
```

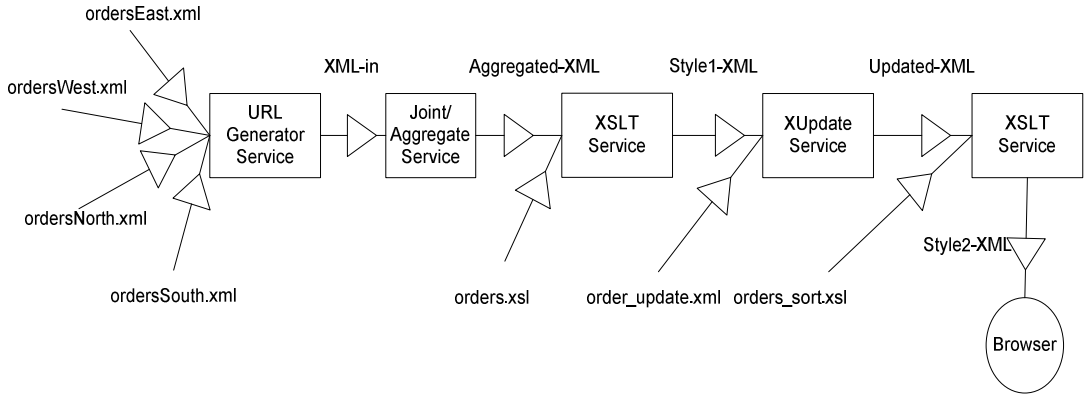
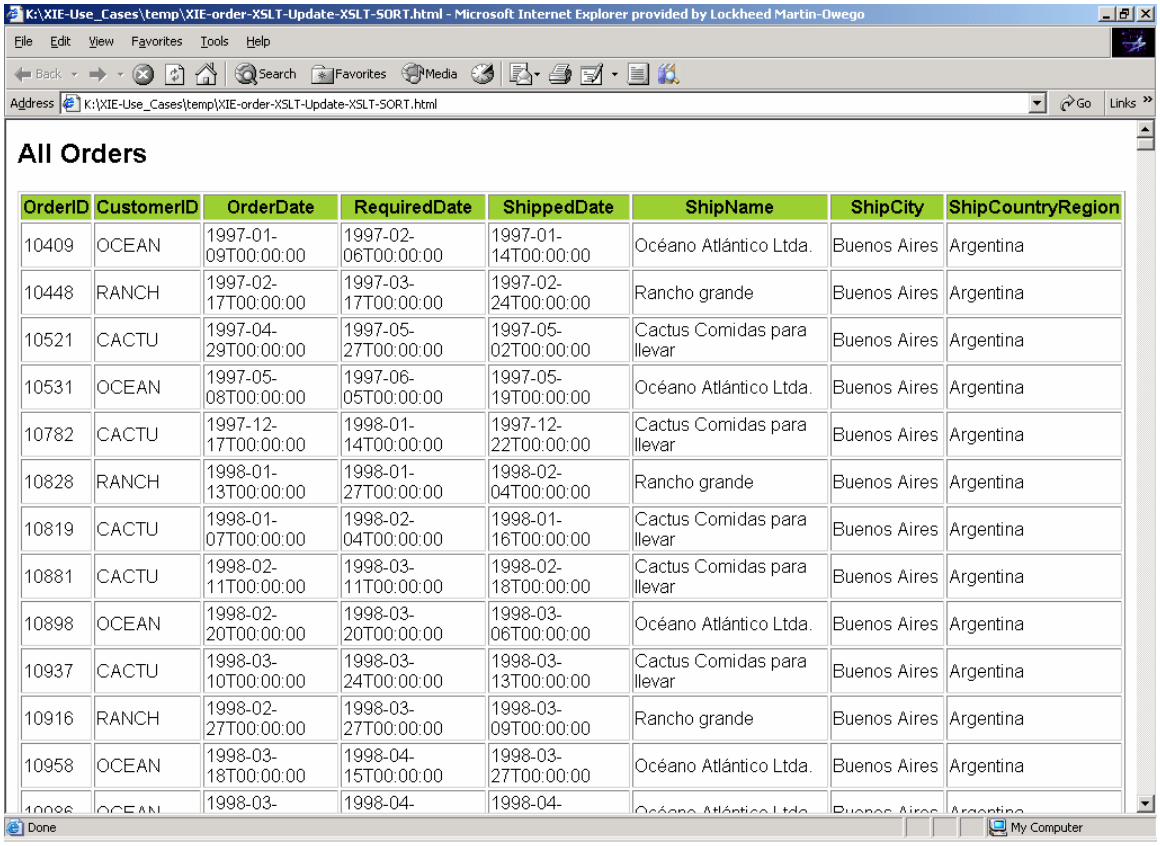


Figure 16: An Order Processing Pipeline

Here are the execution steps for the order processing pipeline:

1. The ordersEast.xml, ordersWest.xml, ordersNorth and ordersSouth input documents are read by the URL generator service. The generated outputs are passed along as inputs to the Join/Aggregate Service.
2. The Join/Aggregate service executes: data from all regions are aggregated in one root element called allOrders. The aggregated output is passed along as an input to the XSLT service.

3. The orders.xsl style sheet is read and executed: the OrderID, CustomerID, OrderDate, RequiredDate, ShippedDate, ShipName, ShipCity and the ShipCountryRegion elements are selected from the orders aggregated input.
4. The selected data is passed to the XUpdate service.
5. The order_update.xml update program is read and the XUpdate service executes: the ShipCity element for OrderID 10716 is updated to Dayton.
6. The updated data is passed to the second XSLT service.
7. The orders_sort.xsl style sheet is read and executed by the second XSLT service: the elements are sorted on the ShipCountryRegion, ShipCity and RequiredDate elements.
8. The sorted orders data is passed along and displayed by the browser as depicted in Figure 17 below.



OrderID	CustomerID	OrderDate	RequiredDate	ShippedDate	ShipName	ShipCity	ShipCountryRegion
10409	OCEAN	1997-01-09T00:00:00	1997-02-06T00:00:00	1997-01-14T00:00:00	Océano Atlántico Ltda.	Buenos Aires	Argentina
10448	RANCH	1997-02-17T00:00:00	1997-03-17T00:00:00	1997-02-24T00:00:00	Rancho grande	Buenos Aires	Argentina
10521	CACTU	1997-04-29T00:00:00	1997-05-27T00:00:00	1997-05-02T00:00:00	Cactus Comidas para llevar	Buenos Aires	Argentina
10531	OCEAN	1997-05-08T00:00:00	1997-06-05T00:00:00	1997-05-19T00:00:00	Océano Atlántico Ltda.	Buenos Aires	Argentina
10782	CACTU	1997-12-17T00:00:00	1998-01-14T00:00:00	1997-12-22T00:00:00	Cactus Comidas para llevar	Buenos Aires	Argentina
10828	RANCH	1998-01-13T00:00:00	1998-01-27T00:00:00	1998-02-04T00:00:00	Rancho grande	Buenos Aires	Argentina
10819	CACTU	1998-01-07T00:00:00	1998-02-04T00:00:00	1998-01-16T00:00:00	Cactus Comidas para llevar	Buenos Aires	Argentina
10881	CACTU	1998-02-11T00:00:00	1998-03-11T00:00:00	1998-02-18T00:00:00	Cactus Comidas para llevar	Buenos Aires	Argentina
10898	OCEAN	1998-02-20T00:00:00	1998-03-20T00:00:00	1998-03-06T00:00:00	Océano Atlántico Ltda.	Buenos Aires	Argentina
10937	CACTU	1998-03-10T00:00:00	1998-03-24T00:00:00	1998-03-13T00:00:00	Cactus Comidas para llevar	Buenos Aires	Argentina
10916	RANCH	1998-02-27T00:00:00	1998-03-27T00:00:00	1998-03-09T00:00:00	Rancho grande	Buenos Aires	Argentina
10958	OCEAN	1998-03-18T00:00:00	1998-04-15T00:00:00	1998-03-27T00:00:00	Océano Atlántico Ltda.	Buenos Aires	Argentina
10986	OCEAN	1998-03-	1998-04-	1998-04-	Océano Atlántico Ltda.	Buenos Aires	Argentina

Figure 17: Sorted Orders

Chapter 7
Conclusion

In this research, we proposed XML Integrated Environment (XIE) which is a general-purpose service-oriented architecture for processing XML documents in a scalable and efficient fashion. The XIE supports a new software service model that provides a proper abstraction to describe a service and divide it into four components: structure, connection, interface and logic. We also proposed and implemented XIE Service Language (XIESL) that can capture the creation and maintenance of the XML processes and the data flow specified by the user and then orchestrates the interactions between different XIE services. Moreover, XIESL manages the complexity of XML processing by implementing an XML processing pipeline that enables better management, control, interpretation and presentation of the XML data even for non-professional users.

Our completed research works and contributions are summarized here:

- Designed and built a general-purpose XML system that can process schemaless XML data.
- Provided the foundation for a powerful user-driven data management experience by offering tools and constructs that can fully utilize the XML processing power

embedded in the XIE's unified framework and service-oriented architecture (e.g., transformation, XML updates, repeat, select, aggregate and route services).

- Specified, developed and implemented a unified service-oriented layered architecture for processing schemaless XML documents in a scalable, efficient and powerful fashion through the incorporation and use of an internal caching mechanism.
- Proposed, designed and implemented a new service model called Meta-Service Model (MSM) that builds the proper abstractions to describe a service that can be easily mapped to the different tiers of the XIE architecture.
- Proposed, developed and implemented a declarative language called XML Integrated Environment Services Language (XIESL) that described and specified the processing of schemaless XML data. XIESL manages the complexity of XML processing by implementing an XML processing pipeline that enables better management, control, interpretation and presentation of the XML data.

7.1 Future Works

The work presented in this dissertation can be extended by adding new tools. The main extension is developing a graphical XML data flow designer that is user-friendly. The designer should be capable of building, executing and debugging XML data flows using XIESL; incorporating and building more core software service capabilities (e.g. supports for XML schema, compression and encryption); incorporating and building more XIESL core capabilities (e.g. web-service integration); and incorporating a more powerful internal caching mechanism for efficient use and greater flexibility.

References

[1] W3C Recommendation, "Extensible Markup Language (XML) 1.0 (Third Edition)," 2004, *http://www.w3.org/TR/2004/REC-xml-20040204*

[2] W3C Recommendation, "XSL Transformation (XSLT) Version 1.0," 1999, http://www.w3.org/TR/xslt.

[3] W3C Recommendation, "XML Pipeline Definition Language Version 1.0," 2002, *http://www.w3.org/TR/xml-pipeline.*

[4] W3C Submission, "XML Pipeline Language (XPL) Version 1.0 (Draft)," 2005, http://www.w3.org/Submission/xpl/

[5] W3C Candidate Recommendation, "Web Service Description Language (WSDL) Version 2.0 Part 1: Core Language," 2006, *http://www.w3.org/TR/2006/CR-wsdl20-20060106/*

[6] NG Specification, "Relax NG committee," 2001, *http://www.relaxng.org/spec-20011203.html*

[7] D. Draper, A. HalLevy, and D. Weld, "The Nimble XML Data Integration System," IEEE Proc. of Int'l Conf. on Data Engineering, 2001, pp. 155-160

[8] An Open-source XML Publishing Framework, *http://cocoon.apache.org/*

[9] S. Abiteboul, D. Suciu, and P. Buneman, *Data on the Web: From Relations to Semistructured Data and XML*, Morgan Kaufmann, 1999.

[10] J. Hunter and B. McLaughlin, "Easy Java/XML integration with JDOM," Java World Newsletters, 2000.

[11] Sun Microsystems, Java Enterprise Edition Home page, *http://java.sun.com/javaee.*

[12] N. Bieberstein et al., "Service-Oriented Architecture Compass: Business Value, Planning, and Enterprise Roadmap," IBM Press, 2006.

[13] W3 schools training, XSLT – Transformation Home page, *http://www.w3schools.com/xsl/xsl_transformation.asp*

[14] M. Humphrey, G. Wasson, J. Gawor, et al., "State and Events for Web Services: A Comparison of Five WS-Resource Framework and WS-Notification Implementations," Proc. of the 14th IEEE International Symposium on High Performance Distributed Computing, 2005.

[15] Organization for the Advancement of Structured Information Standards (OASIS), "Web Services Security: SOAP Message Security 1.0," available at http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wss, 2004.

[16] F. Cabrera, et al., "Web Services Coordination (WS-Coordination)," August, 2005, http://specs.xmlsoap.org/ws/2004/10/wscoor/wscoor.pdf

[17] CBDI Journal Modeling for SOA, February, 2002, www.cbdiforum.com

[18] M. Champion, C. Ferris, E. Newcomer, and D. Orchard, "Web Services Architecture W3C Group Note", February, 2004, www.w3.org/TR/ws-arch/

[19] F. Curbera, Y. Goland, J. Klein, F. Leyman, D. Roller, S. Thatte, and S. Weerawarana, "Business Process Execution Language for Web Services (BPEL4WS)1.1," February, 2005, http://www.ibm.com/developerworks/library/ws-bpel.

[20] J. Goepfert, M. Whalen, "An Evolutionary View of Software as a Service," IDC White paper, 2002, www.idc.com

[21] M. Shivaram, "Securing Web Services – Concepts, Standards, and requirements," SUN White paper, 2003, www.sun.com

[22] P. Brittenham, "Web Services Development Concepts (WSDC 1.0)," IBM White paper, 2001, www.ibm.com

[23] F. Leymann, "Web Services: Distributed Applications without Limits - An Outline," Proc. of Database Systems for Business, Technology and Web, 2003.

[24] M. P. Papazoglou and J. Yang, "Design, Methodology for Web Services and Business Processes," Proc. of the 3rd VLDB-TES Workshop, Hong-Kong, 2002.

[25] M. P. Papazoglou and D. Georgakopoulos, "Service Oriented Computing," Communications of the ACM, October, 2003.

[26] UDDI.org, *UDDI Technical White paper*, 2001, http: //www.uddi.org/

[27] M. Altmel and M. J. Franklin, "Efficient Filtering of XML Documents for Selective Dissemination of Information," Proc. of the 26th VLDB Conf., 2000, pp. 53-64.

[28] R. Baeza-Yates and G. Navarro, "Integrating Contents and Structure in Text Retrieval," ACM SIGMOD Record, 25(1), 1996, pp. 67-79

[29] T. M. Chester, "Cross-Platform Integration with XML and SOAP," IT Pro, 3(5), 2001, pp. 26-34.

[30] C. Chan, P. Felber, M. Garofalakis, and R. Rastogi, "Efficient Filtering of XML Documents with XPATH Expressions." Proc. of Int'l Conf. On Data Engineering, 2002, pp. 235-244.

[31] H. Lie and J. Saarela, "Multipurpose Web Publishing Using HTML, XML, AND CSS," Communications of the ACM, 42(10), 1999, pp. 95-101.

[32] L. Fegaras and R. Elmasri, “Query Engines for Web-Accessible XML Data,” Proc. of the 27th VLDB Conf., 2001, pp. 251-260.

[33] M. J. Hudson, “XML@Work,” EAI Journal, 2001, pp. 32-37.

[34] A. Bouguettaya, B. Benatallah, L. Hendra, J. Beard, K. Smith, and M. Ouzzani, "World Wide Database – Integrating the Web, CORBA and Databases," Proc. of ACM SIGMOD Conf., 1999, pp. 594-596.

[35] T. Usdin and T. Graham, “XML: Not a Silver Bullet, But a Great Pipe Wrench,” ACM Standard View, 6(3), 1998, pp. 125-132.

[36] C. White, L. Quin, and L. Burman, *Mastering XML Premium Edition*, Sybex, 2001.

[37] T.W. Leung, *Professional XML Development with Apache Tools: Xerces, Xalan, FOP, Cocoon, Axis, xindice*, Wrox Press, 2004.

[38] *The XML Family*, SkillSoft Press, 2003

[39] M. Akif, *Java XML Programmer’s Reference*, Apress, 2003.

[40] M. Jasnowski, *Java, XML, and Web Services Bible*, John Wiley & Sons, 2002.

[41] D. Cheung, S.D. Lee, T. Lee, W. Song, and C.J. Tan, “Distributed and Scalable XML Document Processing Architecture for E-Commerce Systems,” Proc. of Int’l Workshop on Advanced Issues of E-Commerce and Web-based Information Systems (WECWIS'00). San Jose, 2000.

[42] L. Seligman and A. Rosenthal, “XML’s Impact on Databases and Data Sharing,” IEEE Data Eng., pp. 59-67, 2001.

[43] L. Marian, S. Abiteboul, G. Cobena, and L. Mignet, “Change-Centric Management of Versions in an XML Warehouse,” Proc. of the 27th VLDB Conf., 2001, pp. 581-590.

[44] "XUPDATE Specification," 2000, http://www.xmldb.org/xupdate

[45] I. Tatarinov, Z. G. Ives, A. Y. Halvey, and D. S. Weld, "Updating XML," Proc. of ACM SIGMOD Conf., 2001, pp. 413-424.

[46] C. Zhang, J. Naughton, D. DeWitt, Q. Luo, and G. Lohman, "On Supporting Containment Queries in Relational Database Management Systems," Proc. of ACM SIGMOD Conf., 2001, pp. 425-436.

[47] P. M. Tolani and J. R. Harista, "XGRIND: A Query-friendly XML Compressor," Proc of Int'l Conf. on Data Enginerring, 2002, pp. 225-234.

[48] P. Felber, C. Chan, and R. Rastogi, "Scalable Filtering of XML Data for Web Services," IEEE Internet Computing, 2003, pp.49-57.

[49] A. Morishima, S. Koizumi, H. Kitagawa, and S. Takano, "Enabling End-users to Construct Data-intensive Web-sites from XML Repositories: An Example-based Approach," Proc. of the 27th VLDB Conf., 2001, pp. 703-704.

[50] S. S. Chawathe, T. Baby, and J. Yeo, "VQBD: Exploring Semistructured Data," Proc. of ACM SIGMOD Conf., 2001, pp. 603-604.

[51] D. Braga, A. Campi, S. Ceri, M. Klemettinen, and P. L. Lanzi, "A Tool for Extracting XML Association Rules." Proc. of Int'l Conf. on Tools with Artificial Intelligence, 2002, pp. 57-64.

[52] D. Kha, M. Yoshikawa, and S. Uemura, "An XML Indexing Structure with Relative Region Coordinate," Proc. of Int'l Conf. on Data Engineering, 2001, pp.313-320.

[53] A. Aboulnaga, A. R. Alameldeen, and J. F. Naughton, "Estimating the Selectivity of XML Path Expressions for Internet Scale Applications," Proc. VLDB Conf., 2001, pp. 591-600.

[54] www.softcare.com, “What is BPEL and Why Is It Important for My business,” 2004, http://www.software.com/whitepapers/wp_whatis_bpel.php

[55] XML:DB initiative for XML Databases, http://xmldb-org.sourceforge.net/