# Declarative Rule-based Integration and Mediation for XML Data in Web Service-based Software Architectures

Yaoling Zhu

A dissertation submitted in fulfillment of the requirement for the award of

Master of Science (M.Sc.)

to the

Dublin City University

School of Computing

**Supervisor**: Dr. Claus Pahl

January 2007

## Declaration

I hereby certify that this material, which I now submit for assessment on the programme of study leading to the award of M.Sc. is entirely my own work and has not been taken from the work of others save and to the extent of my work.

Signed: 

*ID No.: 52166261*

Date: January 29, 2007

## Abstract

The Application Service Provider (ASP) has started to use Web services to expose data sources and adopted Service Oriented Architecture (SOA) to build data integration infrastructure. XML data integration and mediation in SOA is a complex task. The existing mediation technologies and commercial tools take XSLT as the standard to transform and merge XML documents from various Web services with the intension to deliver a unified view of data. As the number of involved data Web services increases, the XSLT transformation programs lead to poor modifiability and are difficult to reuse.

After a systematic evaluation on the existing XML query and transformation languages based on the defined selection criteria, we propose a services oriented architecture in which a declarative rule-based technique is introduced for XML data transformation. At design time, we reuse and adapt Xcerpt as our integration language that represents integration rules in a declarative and reusable manner. At runtime, the mediated software architecture provides the support of automatically generating the definition and the construction of the service connectors that mediate data from various Web services involved in integration flows. We found through working examples that the software architecture improves the modifiability of the data integration architecture.

# Table of Contents

# List of Figures

# List of Tables

# Chapter 1
# Introduction

Web services and Service Oriented Architecture (SOA) has provided a unified way to expose data sources as services (Alonso et al. 2004). The use of standard technologies reduces data heterogeneity and interoperability and is therefore the key to facilitating data integration (Abiteboul, Benjelloun, and Milo 2002, Zhu et al. 2004). Many organizations have started to adopt SOA using XML-based Web services as the standard infrastructure to integrate heterogeneous and autonomous data sources. However, recent research activities in Web services technology have focused on Web services composition and integration rather than data aspects (Haller et al. 2005, Kavantzas, Burdett, and Ritzinger 2004).

In the Web services context, the data in XML representation retrieved from the individual Web services needs to be merged and transformed to meet the integration requirements. Furthermore, XML query and transformation rules that govern the integration may change. Therefore, data mediation programs that facilitate the connection between integration Web services and data Web service providers need to be adjusted or rewritten accordingly. Most current commercial tools have tended to use XSLT (Clark 1999) to mediate differences in data models such as BEA Liquid data for WebLogic and Aqualogic Data Services Platform (Carey 2006) and Oracle WS-BPEL Process Manager (eLib: Oracle WS-BPEL Process Manager 2005).

## 1.1 The application context

The Application Service Provider (ASP) business model, which has been embraced by many companies, promotes the use of software as a service. Information System (IS) outsourcing is defined (Willcocks and Lacify1998) as "the handing over to third party the management of IT and IS infrastructure, resources and/or activities". The ASP takes primary responsibility for managing the software application on its infrastructure. It uses the Internet as the delivery channel between each customer and the primary software application. The ASP maintains the application and ensures that systems and data are available when needed. Handing over the management of corporate information systems to third party application service providers in order to

improve the availability of the systems and reduce costs is changing the ways to manage information and information systems.

Portals, external web sites provided by Application Service Providers (ASP) for their customers, require integration of data that comes from various sources. In order to consume the data, all participants need to understand the data models and representation. The ASP maintains the application, the associated infrastructure, and the data collected from the customer's environment. The ASP needs to deliver a data integration platform at the enterprise level to ensure systems and data are available when needed.

Data integration aims at bringing together various types of data from multiple sources so that it can be accessed, queried, processed and analyzed in an integrated and uniform manner. It is inevitable that large enterprises like ASP use various systems to produce, store, and search their critical data. Therefore, data mediation needs to be addressed because of differences in data models of data sources in this application context. Data mediation has been widely addressed in component-based software development through adaptor and connector approaches (Szyperski 2002, Crnkovic and Larsson 2000).

## 1.2 Research question

The data integration and mediation problems arise when Web services are assembled to deliver a global view of data, more specifically in the case where data from data services needs to be mapped into a consistent global view. In particular, the loosely coupled nature of SOA required dealing with the highly dynamic nature of service compositions. The XML data integration and mediation is a common problem in the composition of collaborating services. However, ASP as the application context in this thesis demonstrates the need to support deployment of Web service technology and SOA beyond toy examples (Sten and Davis 2004). It is a specific, but important area to meet the need of finding solutions to accommodate constant structural changes in data representations. The aim of this thesis is to explore new techniques and methods to improve the modifiability aspect of data integration and mediation in SOA. Modifiability is "about cost of change and is determined by how functionality is divided at the architectural level and by coding techniques with a component at the component level" (Bass, Clements, and Kazman 2003). In particular, there are many

occasions when Web service providers need data mediation according to a global data model. When the business rules change, a smaller amount of code or none gets affected compared to the existing approaches and commercial tools. Therefore, we focus on investigating the following two important questions in this work.

The first research question is the XML data integration and mediation. The goal of this theme is to identify correspondence between the data and the source and those in the global schema and find the most suitable way to express those in a formal, yet modifiable way. Although XSLT has some success in the area of data translation and query answering (Groppe and Bottcher 2003), XSLT is difficult to write and modify for large-scale data integration because of the template rules (Bonifati and Ceri 2000, Boukottaya and Vanoirbeek 2005). Furthermore, developers need to develop new transformation programs even when a small portion of the data representation changes such as the name of attributes. It is difficult to separate the source and target parts of the data mediation rules as well as the filtering constraints. XSLT does work well in terms of transforming data output from one Web service to another in an ad-hoc manner. For example, the XSLT component in ServiceMix (eLib: Apache ServiceMix) that is an Apache implementation of the Enterprise Service Bus (ESB) infrastructure (Chappell 2004) performs the transformation of XML messages. However, XSLT does not support joining XML documents so that source XML documents have to merge into one interim document before the transformation process takes place according to data mappings between an over-arching global schema (or, mediated schema) and the schemas for various data sources. Therefore, XSLT is infeasible to transform XML document in software architectures where many Web services are involved and the modifiability of integration rules are top priorities.

Secondly, it is Web services composition problem. The most common case is that more than one Web services work together to answer user's queries. The output of one Web service is sent back to the integrator and as input to the next Web service in the integration flow. Each Web service in the integration flow typically performs operations that fulfill a part of the data collection as result of the query. All the results will be transformed and merged together before they are sent back to the requestors. Therefore, we investigate if flow-based Web services technologies can be used to assemble Web services in the context of the integration and mediation of

XML data. In addition, we investigate how the answer to the first problem within an overall architecture for the integration and mediation of XML data.

### 1.3 Research approach

The mediated software architecture for Web services described in this thesis employs declarative rule-based approach to address modifiability problems associated with XML data integration and mediation. The aim of our research approach on the representation of the declarative rules is to build solutions on existing XML query and transformation technologies and techniques. We define the criteria based on our problem setting as well as the existing criteria in the literature to evaluate some of existing query and transformation languages in a systematic fashion. We reuse and adapt Xcerpt, a declarative rule-based XML query and transformation language, to address the first problem in the current approaches in Section 1.2. Unlike the current approaches that use complex XSLT programs to address data transformation problem, the data integration rules in our architecture are specified in a declarative and reusable format. The query part and the construction part of data integration rules are represented separately so that the changes on data integration rules that affects on other rules can be limited. In our proposed software architecture, we have proposed some extensions to the language so that we can build connectors as Xcerpt query programs to connect various Web service providers in the integration flows.

We have identified that the schema integration cannot be fully automated in data integration architecture because the syntactic representation of schemas and data do not completely convey the semantics of different data sources. As a result, for both schema mapping and schema integration, we must rely on an outside source to provide some information about how different schemas (and data) correspond. For instance, a customer can be identified in the configuration management repository by a unique customer identifier; or, the same customer may be identified in the problem management repository by a combination of a service support identifier and its geographical location. Therefore, the focus of our study is not on the automatic composition of Web services, but rather on how the data output from various Web services can be automatically integrated according to a global data model. Therefore, we adopt the Web services Business Process Execution Language (WS-BPEL) (Andrews et al. 2003) as the service composition language in the mediators for the assembly of Web service data providers. The mediator Web services deliver a global

data model built on top of individual data models in dynamic, heterogeneous and open environments.

## 1.4 Thesis contribution

This thesis proposes a declarative rule-based approach that addresses the often-neglected data integration, mediation, and adaptive aspects of service-oriented architecture (Zhu and Pahl 2006b). Our contribution is to provide a Web service-based SOA architecture to address the data integration and mediation problems. Within the software architecture, we have developed a XML data integration and mediation technique by reusing and adapting Xcerpt. There are the following three components in our contribution. Firstly, we define a set of criteria for the selection process for the most suitable language in our problem context. We have determined that Xcerpt can be adapted to be our integration language after a systematic evaluation on a selection of language candidates following these criteria. Secondly, we propose a XML data integration and mediation technique through the declarative, rule-based definition and construction of Web service connectors. Using the declarative rule-based connectors, the modifiability of data integration rules can be improved because the connectors are separate from the integration flow in the mediator Web services. The separation makes it easier to maintain the loosely coupled nature of SOA in the context of XML data integration. Thirdly, we propose a mediator architecture that enables adaptive information service integration based on adaptive connectors.

We believe that this work can be adopted into other problem domains that need data integration and mediation layer to consolidate their data sources. For example, our proposed architecture can work as the mediation layer in the promising ESB technology stack.

## 1.5 Thesis structure

This thesis is organized as follows. In Chapter 1, we present the motivation of this work, the research question, and the research approach and thesis contributions. In Chapter 2, we discuss the background of the data integration domain and traditional architectural solutions. In Chapter 3, we appraise the state of the art and related research, and present the principles of our declarative data mediation technique. In Chapter 4, we investigate the selection criteria of a suitable XML query and

transformation languages in the literature and identify our own criteria to address the data mediation problems. In Chapter 5, we outline the proposed software architecture and the limitations of the chosen language in order to make it to fit into our overall architecture. In addition, we extend the chosen language including the resource specification and querying the metadata. In Chapter 6, we follow the Architecture-level modifiability analysis technique to evaluate the architecture and present our results. Finally, we summarize our contributions and outline the future research directions in relation to this work in Chapter 7.

# Chapter 2
# The background

Data integration (also called information integration) is the problem of combing heterogeneous data residing at various sources, and providing users with a unified view (Lenzerini 2002). The unified view is represented by the global schema in an attempt to reconcile all data sources to answer users' queries. Data mediation problems arise because of difference in data models. The goal of a data mediation approach is to find a way to express and query the correspondence rules more efficiently and even automatically (Garcia-Molina et. al.1997).

In Section 2.1, we discuss the concept, the problems, and associated tasks of data integration from the data aspect. The characteristic of data that needs integration play a key role on deciding which the integration approach is more suitable for them. We start with data integration in the traditional database management domain; followed by data mediation challenges that XML and semistructured data based data integration systems face. We finish this section with the characteristics of the XML data in the Web service-based systems. In Section 2.2, we discuss the two major architectural approaches in the literature related to the data integration: the data warehousing approach specializing in applications where the response time is the dominating quality attribute and generating highly analytic reports and the mediated approach specializing in applications where always having the live data is the dominating quality attribute.

## 2.1 Data integration

Many Web applications such as portal, e-commerce applications have the mandate to reuse the data sources of legacy applications to support new business processes. A data integration process typical starts with client applications strive to identify data sources that provide the data to support new business processes. Developers have to understand the data sources and the access mechanism. Possibly a dedicated data access method or adapter is needed to get the data. Having realized the advantage of a global data model, the request of having a data integration framework arises in order to support the global data model. At design time, business analysts working on Web applications start with trying to understand and absorb how the data can be

consumed and viewed. The goal of data integration framework is to provide a common interface to a multitude of data sources, despite the data models and data representation of them (Levy 1998). In addition, passing the data in and out of the data sources needs data mediation because the global elements in the global data model may have various elements of the same thing in the data sources.

### 2.1.1 Traditional database data integration

The data integration problems in relational databases have been studied extensively in recent years (Sheth and Larson 1990, Abiteboul 1997, Levy 1998). In the traditional database management systems, data integration is the process of the standardization of data definitions and data structures by using a common conceptual schema for various data sources (Heimbigner and McLeod 1985). Sheth and Larson (1990) introduced federated database systems to deal with distributed data integration in a network.

The challenge that federated database system face limits to dealing with the distributed, heterogeneous, and autonomous data. In terms of distributed nature of the data, communication protocols are use to connect data sources such as TCP/IP and HTTP. In terms of heterogeneous nature of the data, there are many forms of heterogeneous, such as structure, constraints, the query languages, and even semantics (Sheth and Larson 1990). One of the tasks to address data mediation problems is to define the data mappings between the individual data sources and the global view of these sources because of differences in data models. However, on the data sources side, not all data of data sources is available to the federated schema. Therefore, an exported data schema is created specifically for the federated schema built on top of data sources. The export schema contains the real data while the federated schema provides a reconciled, integrated, and virtual view of the underlying sources. Most of the research on this area focuses on structural heterogeneities and semantic that arises upon mediation.

### 2.1.2 XML-based data integration

The Internet and the Web has changed the distribution and representation of documents for exchange, consequently the scope of data mediation has been extended to integrate the semistructured data sources such as XML/HTML documents. The semistructured data is in contrast to the structured data stored in the

rational database systems. The semistructured data may not have a schema definition (Abiteboul 1997 and Abiteboul, Buneman, and Suciu 2000). The key to the integration of semistructured data is to have a canonical data model. One of the most important data model for the semistructured data is Object Exchange Model (OEM) proposed by the TIMMS system at Stanford University (Abiteboul et al. 1997). The TIMMIS system defines objects in the OEM data model as "an OEM object is a quadruple (label, oid, type, value), where label is a character string, oid is the object's identifier, and type is either complex or some identifier denoting an atomic type (like integer, string, gif-image, etc.)". Most systems for data integration use a self-describing nested data model to deal with the unpredictable, unstructured information like OEM and its variations.

As the emergence of XML as the standard data on the Web and database systems, the structural heterogeneous side of the data has been faded. The structure of XML documents is simpler and flexible as it provides a standard representation of the data because XML documents are often with a schema definition such as XML Schema (Thompson et al. 2001). XML Schema provides enough expressive power to represent the data on the Web and the Web applications. Nowadays, Web applications produce XML outputs as well as HTML outputs via the APIs to access the data sources behind the web applications. However, the schema integration is still one of the main tasks involved as Lenzerini (2002) argues that data integration is the problem of combining data residing at various data sources, and providing the user with a global view of these data.

### 2.1.3 Web service-based data integration

In Web service-based data integration systems, data exchange and integration solely relies on XML. Therefore, we define the global schema with the XML schema definition in this thesis. There is no need to build wrappers around the data sources since all the data are encoded to conform to the type definitions in the XML schema. Rather than the data residing at different data sources, the data resides in the data sources behind Web services. In addition, data integration systems need to maintain its "loosely coupled" nature of Web services architecture by writing as less code as possible in term of Web services assembly and mediation. Although open standard interfaces that Web services technologies provide have replaced the traditional wrappers, there is still need to be able to find data that they want just in the same

way as traditional data integration systems. Beside the integration and mediation of the XML data, data integration systems may have requirements to perform the update operations on the global schema that passes on the updates back to data sources wrapped behind Web services interfaces. Therefore, we need ways to build a conceptual integration flow to deliver the global data model. Milanovic and Malek (2004) have conducted a survey of approaches towards Web services composition evaluated.

**Semantic Web approach**. The Semantic Web approach enables automatic services invocation and composition (Haller et al. 2005). In composition time, the domain experts identify the correspondence between the global schema and the Web services. The correspondence is specified by either domain ontology or the model transformation rules (Gruninger and Lee 2002). When the user's requests come in, the integration framework looks up ontology repository to generate an integration plan for the requests automatically. However, the automatically generated integration plans are often inefficient (Thakkar, Ambite, and Knoblock 2005). Some optimization techniques are needed after the generation to make to improve its execution time. Authors in (Lehti and Fankhauser 2004) proposed to use OWL as extensions to XQuery (Katz 2004) to validate the consistency of these correspondences.

**Web components**. The web component approach uses Component-based Software Engineering (CBSE) to model Web services as components to aims to increase reliability and maintainability (Szyperski 2002, Crnkovic and Larsson 2000). Efforts have already been made to make the Web suitable for the discovery and usage of software applications instead of documents. These efforts are bundled in the Web services architecture (Booth et al. 2002). However, software components on the Web are more than a collection of services. Component development in a distributed environment such as the Web requires precise information about components and their services (Pahl 2002). Web component framework has been proposed to enhance the Web Services Framework and allow successful software engineering technologies to be utilized on the Web and using Web technologies (Pahl and Zhu 2005).

**WS-BPEL approach.** Many workflow-based languages such as WS-BPEL, Web Service Flow Language (WSFL) (Leymann 2001) and WSCL (Banerji et al. 2002)

have been proposed to handle the workflow within the global XML schema. WS-BPEL and other workflow-based XML standards provide an integration model based on Web service interactions among the involved web service providers in the form of business processes. Authors in (King and Roantree 2005) propose an ontology-based framework where the WS-BPEL process flow can be semantically constructed based on the semantic descriptions of Web services represented with the OWL-S ontology language (Martin et al. 2004).

**Distributed query engine approach**. At design time, developers define XML data services by creating query programs to specify correspondence between data sources and the global schema in XML query languages such as XQuery. At runtime, the user's requests will be translated into XQuery queries that are passed in the XQuery engine to generate the execution plan on individual Web services. All the web services are considered as XML data sources in this approach. The drawback of this approach is that the data sources are very tightly coupled to the XML data services. BEA Liquid data for WebLogic and BEA Aqualogic Data Services Platform (Carey 2006) take this approach.

A WS-BPEL process flow is itself a web service serving in an intermediary role between service requestors and service providers. Traditional data integration systems require data mediation and decompose query requests across the multiple sources. As update operations are not required in our problem context, we argue that the data integration and mediation component is better off being separated from the query decomposition flow. Therefore, WS-BPEL can be used as our service orchestration language although it is primarily designed for the business processes and applications integration. However, the service orchestration to assemble Web services alone cannot provide XML data integration and mediation in SOA. XSLT transformation programs used in traditional data mediation approaches only provide a very simplified level of data transformation.

However, the mediation for XML data remains to be solved. The global XML schema consists of the integrated objects (or views) that are constructed from the selection of the value of attributes from the data sources. After XML documents returned from the web service data providers, they need to be merged together to construct new XML documents conforming to the global XML schema. XML document mentioned in this thesis is the payload of the Simple Object Access

Protocol (SOAP) messages. The most common approach among the industries and research solutions is to code up these transformation rules in XSLT and inject the XSLT transformation files into the WS-BPEL process flow. There are a few advantages of choosing XSLT, for example, it is a standard transformation language recommended by W3C and well supported by the industry. However, there are some disadvantages in this approach. For example, the mix of query parts and construction parts, a minor change on the rules affects the entire XSLT program. We can imagine that the amount of labor intensive work has to be carried out to rewrite the XSLT programs as the changes of the global XML schema definitions and the local XML schema definitions, it is almost impossible to maintain these transformation rules within the XSLT scripts. In addition, integration at the data or information level is tightly coupled with the process flow and therefore, modifiability of this data integration will be expensive in the long term. If the data transformation rules have to be identified by a human user, we argue that the generation of the query programs, which can be executed on the source XML document, will be a big step forward to increase the productivity.

## 2.2 Approaches to XML-based data integration

There are two major approaches to the data integration problem (Widom 1995). One is the data warehousing approach (is also called the federated database approach). The data warehousing approach extracts data from the export schema exposed by data sources in advance according to the integrated view and materializes them in databases. The second is mediated approach. The mediated approach gathers data from the appropriate data sources at runtime to populate the entities in the unified virtual view. In the former approach, queries imposed on the integrated view are answered directly from the materialized view. In contrast, queries imposed on the integrated virtual view in the latter approach are decomposed into the sub-queries that will be answered at the individual data sources level. Widom (1995) also identified the circumstance the approaches suited best respectively. In the following subsections, we will outline the details about these two approaches and an analysis the differences, and the application contexts.

### 2.2.1 Data warehousing approach

A data warehousing approach to integration is suitable for data consumers wanting to access to local copies of the data so that it can be modified and calculated to suit the business needs by nature, and the query performance is the vital factor for the system. Data warehouse systems may contain many views to meet the business needs largely for reporting purpose. Therefore, the aggregation of data and the analytic data based on the data sources are also stored in the data warehouse. In basic data warehouse architecture, the data sources are connected to the integrator via the wrappers. Wrappers have the responsibility for translating the data format into the format and the data model used in the data warehouse. When a new data source is added into the system, or the changes may occur on a data source, the new or modified data is pulled back into the integrator. The integrator is responsible for gathering information in the system, such as filtering out the irrelevant information, merging information from the data sources together according the data model in the system. We illustrate the relationships among components in mediated architecture in Fig. 2-1.

In most cases, the data to be integrated is owned and maintained by many IT departments within an organization, therefore, it is uneasy to build another data source in additional to the existing data sources.

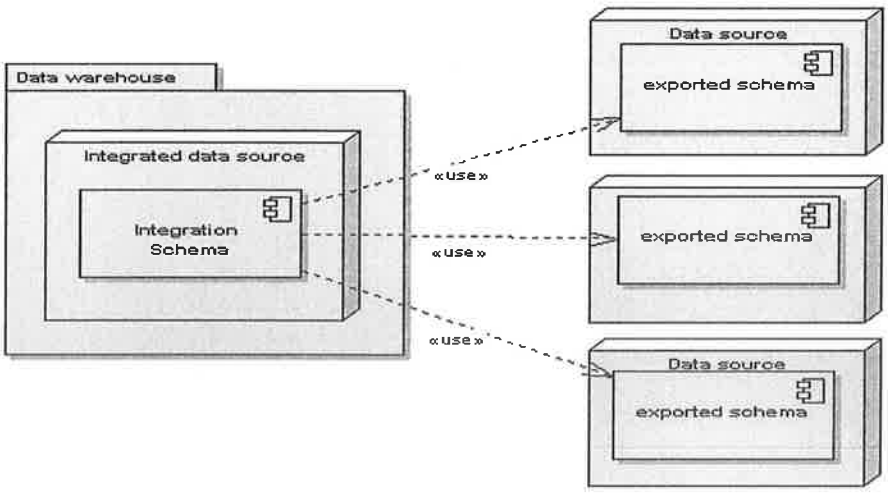


Fig. 2-1 The architecture of a data warehouse system.

### 2.2.2 Data mediation approaches

The traditional data mediation approaches focus on the data with no schema attached to it and the semistructured data. The customers can access the data sources via the

wrappers, which build around the heterogeneous data sources; they also may access the data via the mediators. Queries posted on the mediators need to reformat into sub-queries that are executed on the wrappers of the data sources. In addition, the data brought back to the mediators will go through a dedicated data merge engine to return the refined data sets conforming to the common and consistent data model back the customers.

The data mediation approach to integration is suitable for information and service environment that changes rapidly, and for queries that operate over large amount of data from numerous information sources and most importantly for clients with the need of the most recent state of data. Wiederhold (1992) identifies the need for mediators that harmonize and present the information available in heterogeneous data sources. This harmonization comes in the form of identification of semantic similarities in data while masking their syntactic differences. Mediators enable relevant and related data to be integrated and presented at a higher layer of applications in Fig. 2-2. The mediators will hide the underlying communication protocols and the detailed implementation from the developers. Data mediation is a central architectural composition aspect. The main tasks involved are to build up wrappers around the unstructured data sources as the interfaces to interpret the common query language operated at the mediator level. The wrapper layer also translates the query requests into the query languages can be accepted in the data sources. Mediator concepts have inspired many data mediation architectures built such as TIMMIS, Garlic (Carey et al. 1995) and Information Manifold (Levy 1998).

Garcia-Molina et al. (1997) identified the following requirements are essential in order to build mediator architecture. Firstly, it has a global data model that is more flexible than the models commonly used for the database management systems. Secondly, it has a common query language. Thirdly, a tool makes the creation of a new mediators and mediator systems. The two approaches in the literature have been proposed to specify correspondences between data sources and the integrated schema (Ullman 1997). One is Global-as-View (GAV) (Papakonstantinou, Garcia-Molina, and Ullman 1996) and Local-as-View (LAV) (Levy 1998). The GAV approach defines the entities in the global data model as views over the export schemas whereas the latter approach defines the export schemas as views over the global data model. Both of two approaches answer users' queries using views. In GAV

approach, the correspondence tells the system how and where to retrieve data explicitly. This approach is effective when data sources are reliable and stable. In contrast, the LAV approach tries to address adding new sources into the data integration system because data sources are not related to each other. Quite similar to the LAV approach, Hasselbring (2002) proposed the yo-yo approach in which the domain models are built to begin with the common domain-specific standards. However, there could be many domain models in the yo-yo approach as apposed to the single global data model in traditional data integration approaches. The domain models in this top-down approach are independent to the data models of legacy systems because the component models are introduced to interact with both sides acting as the intermediary.

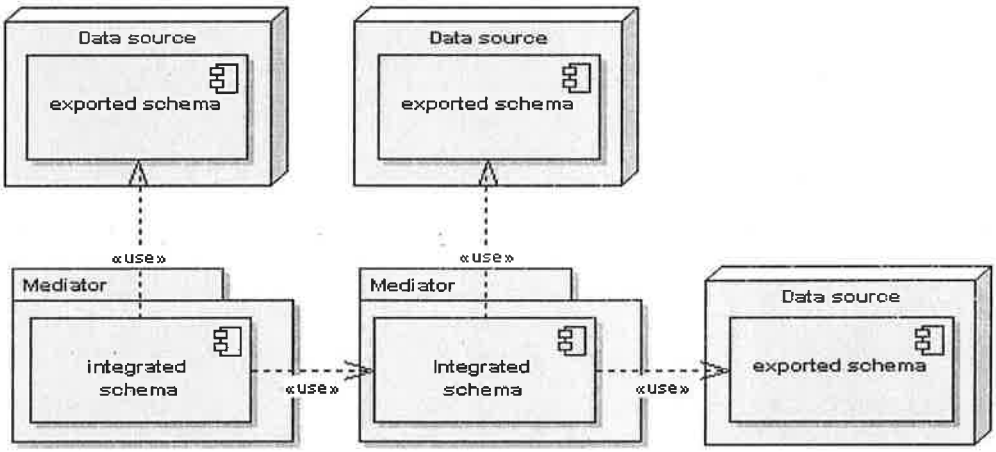


Fig. 2-2 The mediated architecture.

The schema definition of both the integrated view and underlining data sources are mandatory for an integrated view no matter which approach the data integration system is based on. For the data sources that are not relational databases, the schema definition of a wrapper around the data sources has to be defined in advance. Data mediation in the XML data integration architecture is also called schema integration. XML has been chosen as a middleware model for data integration system because of its flexibility (Papakonstantinou and Velikhov 1999). In the problem context of this thesis, the data that reflects the delivery of services to customers have to be live data. For example, customers have the right to request to view the status of their service requests and the availability status of their services. The data sources are reliable and stable because they are internal to the ASP. Query processing is a difficult task because the global schema does not have any knowledge about the data sources in

the LAV approach. Therefore, in the context of our ASP problem setting, the GAV approach seems to the sensitive choice.

## 2.3 Summary

In this chapter, we have explained the concepts of data integration and problems data integration architectures have to address in both traditional data integration architectures and the Web service-based architecture such as SOA. We have found that the SOA solutions still face the same data mediation and data transformation problems and raise several more problems in order to build up a common data model among Web service providers although architectural changes. In the next chapter, we review the state of the art and related work related to the XML-based data mediation and data transformation approaches in detail.

# Chapter 3
# The state of the art and related Work

This work is motivated in the context of application service providing with service-oriented architectures. However, the research problems and research approaches are in the area of XML transformation within service-oriented architectures with a shared global schema. In this chapter, we discuss the related research approaches. In Section 3.1, we discuss schema-based approaches in the literature in terms of semantic integration and data translation. In Section 3.2, we discuss the model transformation approaches that automate the data transformation processes.

## 3.1 Schema-based XML data integration

The purpose of a data integration system is to integrate a set of existing data sources. Therefore, the first step of an application domain expert is to design a mediated schema. The mediated schema describes the portion of data that data sources may want to share. One of the most important benefits that data integration systems may bring to the business is that it frees up users from the labour-intensive tasks. These tasks include searching for the relevant data sources or trying to understand the data structure of the data sources. As a result, it enables users to focus on the application logic. The mediated schema encapsulates the detail of how to retrieve the answers in terms of interfaces, communication mechanism. The schema integration is the key component that governs the data flows between the source schemas and mediated schema by maintaining the relationships between them. Therefore, among many problems in the data mediation domain, data translation from a multitude of data sources to the data conforming to the global data model continues to be the most important one. In the data warehousing approach, the application domains experts start by identifying those elements in the schemas from the data source are also present in the warehouse. After the initial mapping is created, the transformation needs to be created between the mediated schema and the local schemas.

In a summary, the schema integration process consists of two steps: firstly the creation of the integrated schema and secondly the creation of the correspondences between the integrated schema and the source schemas. This assertion states how the set of values of a construct in one source schema relate to the set of values of a

construct in another source schema. In traditional data integration systems, the process of creating correspondences among schemas consists of coupled steps because the query rewriting and data merging process depends on the correspondences. However, for the Web service-based data integration systems, the creation of the correspondences step has become an explicit process.

There are two approaches to express the inter-correspondences. One is semantic schema matching approach, the other is to use declarative transformation languages express correspondences. Indeed, graphic tools are often used to build up the query programs. The semantic schema matching approach is to use the schema matching techniques to automate the mapping process either by extracting the schema information or by building up the domain ontology. Semantic Web related technologies are one of many ways to explore the semantic similarities of schemas. The core part of schema matching techniques is building up a semantic rich data model such as middleware data model in (Abiteboul, Cluet, and Milo 1997) on top of the local schemas. However, domain experts need to verify and confirm semantic because more than one semantic relationship may be discovered. The declarative transformation language approach is to use a transformation language to express the inter-correspondences among schemas to reduce the step of generating schema matching template as the former approach. In the following subsections, we outline two approaches toward the schema-based data mediation.

### 3.1.1 Semantic integration approach

Schema matching is a schema correlation operation that takes two schemas as input and generates the mapping that expresses the elements correspondences of the two schemas (Madhavan, Bernstein, and Rahm 2001). The goal of schema matching is to automate the process of identifying the correspondences between the integration and the source schemas by semantic meanings of the elements such as foreign key relationships in relational databases and IDREFS in XML schemas. Many attempts of the sort are either semi-automatically or with the support of graphic user interfaces, e.g., Clio (Miller, Haas, and Hernandez 2000), Cupid (Madhavan, Bernstein, and Rahm 2001). A survey of automatic matching techniques and an analysis of the matching approaches are illustrated in (Rahm and Bernstein 2001). The Schema matching approaches normally take the inputs schemas and output the mappings, some of them go further to generate the query program by composing

these mapping which can be carried out on the schema instances. Defining the match operator such as entity-relationship (ER) model needs a common representation for inputting schemas and outputting mappings. Although there are many approaches proposed in this area, semantic integration remains an extremely difficult problem. For a data integration system without a predefined integration schema, in order to extract the semantic s of the involved elements, there are very few sources to look for such as documentations of the data sources and associated schema and data. Therefore, the schema elements are matched merely based on the names and types of the elements at the schema level and the data constraints of the elements at the schema instance level (Rahm and Bernstein 2001). The matching candidates such as schema and the data are often incomplete, and subjective to the application domain presented in (Doan and Halevy 2005). Domain knowledge is the key success to the semantic integration. The more is domain knowledge available, the more accurate is the complex matching. Currently the similarity value between the elements of the source and the target are expressed between zero (dissimilarity) and one (similarity). As a result, a human user needs to choose from a set of mappings generated.

In Clio, value correspondences drive the schema integration. Value correspondence can be achieved by defining how a value in source schemas can be used and which a value in which source schema should be used if multiple values found in the target schema. Schema mappings can also be derived by the most-used queries that have been built by the database administrators. The target schema does not depend for its definition on the schema structure of the sources. The source schemas are normally relational database schema with foreign key relationships enabled among the tables; the target schema can be either relational database schema or XML Schema. The process of deriving schema mapping is largely a manually process due to the human attention. The schema matching in Clio has two phases. Once schema mappings are generated and specified, Clio is able to generate XQuery (Boag 2004) query programs or SQL queries for the actual transformation and integration.

Clio has proposed a new technique to derive schema mappings among schemas. One of main contributions in Clio is to use the most-used queries to represent schema mapping. However, initial value correspondences have to be established in order to create semantic relationships; the mapping can only be established between

one source schema and one target schema at a time. Clio also relies on heavily on the relationships among the elements in the source schema. Therefore, Clio is limited itself to address schema translation between the relational schema language and other schema languages. Furthermore, the application domains of Clio are data warehouse and information systems rather than Web service-based SOA architecture.

Schema mapping tasks are often repetitive. Therefore, machine Learning techniques that taking the some manually created schema mappings as training data are proposed to predict mappings. A machine-learning approach is presented in (Doan, Domingos, and Halevy 2001) namely LSD to derive the semantic mappings between pre-defined integrated schema and the source schemas. First of all, users needs to supply a small portion of the schema mappings from one source schema to the target schema to train the machine learner, the machine leaner can then discover the semantic mappings for the new source schemas added into the system. As the user understands the business domain better, the more types of machine learners can be added into the system to improve the accuracy of the mappings. XML documents with associated schema definitions are the primary target data. The LSD system consists of four components: base learners, meta-learner, prediction converter and constraint handler. The schema matching process is based on the name and data content matching carried by the base learners after feeding them the train data. The predictions generated from the base trainers are then combined in the meta-date learners.

Zhu et al. (2004) proposed a service-oriented architecture to integration remote traditional database with a shared mediated schema. A Web services layer is built on top of traditional databases with each of Web services registered into a private registry. An ontology service component is used to provide data transformation based on the published XML schema of each registered Web services at runtime so that every new Web services can be dynamic discovered and integrated. The schema mappings are derived by domain ontology. Their approach has proved that Web service-based SOA architecture can be used in the data integration domain in large projects. However, the ontology-driven schema mappings are quite complex and often not broad enough to derive all schema mappings.

Xyleme (Reynaud, Sirot, and Vodislav 2001) also takes a semantic mapping-based approach. It uses a tree as the global schema and source schemas are mapped to this tree through path mappings. However, it requires the XML documents are stored in a repository. A recent study proposes an approach of materializing an integrated schema resulting from the transformation of several source XML schemas using AutoMed (Zamboulis and Poulovassilis 2004). They have provided two algorithms, the first for restructuring the schema of an XML document into a target schema, and the second for materializing the global data model resulting from the transformation of several source XML documents. However, the structure of a given XML document cannot be modeled with the XML Schema.

### 3.1.2 Data translation

Data translation by definition is to transform the data for one format into another, or from one data type within a data model into another types within another data model. The data translation is a process focusing on translating the heterogeneous data in one format such as SQL to another format such as XML (Abiteboul, Cluet, and Milo 1997, Cluet et al. 1998). However, just like the schema integration problems, the translation tasks also consists of the derivation of correspondences rules and then the data translation from one format to another format. In data translation systems, the correspondence rules, transformation of data formats can be specified using a single declarative set of rules. The main goal of the data translation the data translation rules are often derived automatically from the correspondence rules.

TranScm (Milo and Zohar 1998) identifies matches between schemas in term of the structural similarities of the components. The structural similarities or differences are expressed in rules. Rules can be used to define a possible matching between two schema components, and provide the correspondences for the later data translation of an instance of the source schema to an instance of the target schema. These rules are predefined in the systems as the data translation templates that may be used to handle the most common translating cases. Users can extend these templates to suit the business needs during the translation process. TranScm is also a two-phased process in terms of data matching: the rules find the best matched common components between the schemas; secondly, and the data translation process starts automatically using the translation functions associated with the rules. The main contribution of TranScm is in the translation functions of the pre-defined matching rules templates.

The using of built-in translation functions can save a large amount of time of developers' time to program the data transformation programs when the matching components between the schemas can be found in the system. However, developers have to program the data components that cannot be handled in the TranScm system.

### 3.2 Model transformation and DSL approach

Model transformation and Domain Specific Language (DSL) (Van Deursen, Klint, and Visser 2000) have been introduced to transform the software meta-model. Model Driven Architecture (MDA) (Frankel 2003) has been proposed to solve the problems arisen during the transformation process such as handcrafting the transformation programs. A study by (Peltier, B´ezivin, and Guillaume 2001) introduces MTRANS language that is placed on top of XSLT to describe model transformations where XSLT is generated from MTrans formalism. MTrans is a two-level framework, the abstract model level and the concrete model level based on the four level architecture of Meta Object Facility (MOF) specification (eLib: OMG's MetaObject Facility). The transformation rules are expressed in the form of formalism, and will be parsed based on a compiler generator. They argued that the transformation rules are best expressed at the abstract model level rather than at the concrete model level to reduce the complexity of the transformation rules. The transformation between metamodels can be automatically carried out. The inputs to the DSL are the source XML Schema definitions and the transformation rules. XSLT is used for the output of the Model translator. However, it only demonstrates an approach of transforming one model to another; it still does not solve the data mediation issues, which the source data coming from local schemas will be integrated into one global schema. In addition, the MTRANS formalism and later MTRANS DSL lacks of the expressive power to query and compose the transformation rules as other languages such as XML-QL (Deutsch et al. 1998).

*fxt* (Functional XML Transformer) is another DSL approach to transform XML documents (Berlea and Seidl 2001). *fxt* is a two-phased process. Firstly, the XML document waiting for transformation needs to be converting into a XML tree structure by Java DOM APIs. Document Object Model (DOM) is a component API of the Java API for XML Processing. The *fxt* transformation specification expressing transformation rules apply on DOM tree to mark nodes found by matching patterns in the specification in the transformation process. Secondly, the fxt transformation

specifications convert founded nodes into certain structures expressed in the selection patterns of the specification. *fxt* is a M2 Model in MDA architecture. *fxt* is a rule-based language to express matching patterns and selection patterns. An fxt transformation specification like XSLT style sheet is compiled into SML (Milner et al. 1997) code. In a summary, fxt is merely a XML transformation language that is suitable for transforming XML documents. It does not have constructing and integrating functionality as existed in the other Query languages; the more complex transformation task has to be carried out by embedding SML code into the transformation specification.

In the model transformation world, there is an ongoing effort standardizing transformation languages led by the OMG. The proposed Query, View, and Transformation (QVT) architecture (OMG 2005) will be capable of expressing queries, views, and transformations over models in the metamodel architecture. The QVT architecture defines three languages that each of them is designed to solve some aspects of the model transformation problems. The *Relation* language of QVT is more a patterns-based language that is capable of expressing data relationships (or data mappings) and traceability among models. The *Core* language of QVT is a declarative model transformation language that can be used to specify the semantics of the *Relation* language at the higher level. The *Operational Mappings* language of QVT extends the Rations languages with the imperative constructs and OCL constructs such as loops and conditions to provide a complement solution for model transformations. Although the QVT languages can be used to specify the data mappings from the data model of each of data sources to the global data model, however, QVT approach is currently not well supported through tools and accessible tutorial material let alone it is only very recent standardization.

### 3.3 Declarative rule-based specification approach

Many data mediation systems adopt logic rules to express the correspondences between the schemas (Levy 1998). In general, logic rules elicit schema information such as element names, schema structures and integrity constraints. The rule-based approach provides the following three advantages compared to other approaches. Firstly, the declarative rules tend to be generic. For example, the rules used to express correspondences between integrated schema and source schemas can be used to build query programs. Secondly, the rules are intuitive to learn for users and

inexpensive to use for the data integration systems. In contrast, matchers in the machine learning approaches need training. Furthermore, one matcher has to work together with other types of matchers in order to cover a bigger area of the schemas. Thirdly, rules are also useful to derive new rules to create new elements in the integrated schema in an automatic manner.

In semistructured data integration systems, the schema integration is expressed by a set of specifications. A specification consists of a rule head and a rule body. A rule head describes a mediated view object; whereas the rule body describes conditions that a source object has to satisfy in order to realize the relationship. A rule is a correspondence between a source object and a mediator object. Evaluating a single rule only populates a fragment of the mediator object. Therefore, all the rules designated to a mediator object can be evaluated incrementally and independently to populate a mediator object. The whole process is called object fusion (Papakonstantinou, Abiteboul, and Garcia-Molina 1996). The object fusion plays a key role in integrating the sources together into mediator objects. A mediator is rendered virtually by a set of rules with the same object id. The object id is semantically meaningful object identifier. The specifications are based on OEM object denoted by (object-id label value). The advantage of object fusion approach is to make adding one more source very easy by introducing one more rule.

MSL (Mediator Specification Language) introduced in MedMaker (Papakonstantinou, Garcia-Molina, and Ullman 1995) is also declarative rule-based language to construct the mediators based on the declarative specifications. XML query languages such as XMAS (Ludoscher, Papakonstantinou, and Velikhov 1999) and XML-QL (Deutsch et al. 1999) were also proposed to query the semistructured data and construct the returned output. The rules or specifications can be queried by MSL (Mediator Specification Language). MSL is powerful enough to carry on operations such as grouping from one source object, removing redundancies and removing inconsistencies. These rules are declarative, rather easy to understand and provide a high level of abstraction. These rules are mainly deduction rules (or short "rules"). A deduction rule is generally an If . . . then . . . statement. If the query pattern in the rule body is satisfied, then the construct pattern in the rule head is assumed to hold. Usually, the construction pattern uses data selected in the query pattern. In a sense, a deduction rule is similar to a VIEW in relational database

systems. These data transformation rules involve selection, extraction, transformation, aggregation and even grouping.

The Business Rules Group (Hay and Kealy 2000) gave a formal definition on business rules, as "a business rule is a statement that defines or constrains some aspect of the business. It is intended to assert business structure or to control the behaviour of the business". Business rules provide the rich knowledge behind the business process and data integration applications. How to elicit the business rules from the business requirements has become the core part of the requirements engineering process. Business rules consist of business statements and policies that are external to the data integration systems. However, they are the key factors that influence how data integration mappings are generated. As a result, the behaviour of data mediation process can be governed by business rules. The business rule-based approach can be applied into the data transformation domain as presented in (Orriens, Yang, and Papazoglou 2003). The data integration engine is built in WS-BPEL, the composition schema are in activity diagram as its business logic and the components invocation orders are pre-defined in the composition schema, orchestrations are defined by specifying which operations to invoke from the beginning of the execution to the end. The business logic in this thesis is defined as the business rules that govern the data integration process. The data integration rules are generally elicited from the business logic. The common information model governs what types of services and components are involved in the composition. A business rule engine based approach has been introduced to separate the business logic from the WS-BPEL process in (Rosenberg and Dustdar 2005).

Xcerpt (Bry and Schaffert 2002a, Schaffert 2004) is a query language designed for querying and transforming both data on the "standard Web" (e.g., XML and HTML data) and data on the Semantic Web (e.g., RDF data). The design principles are given in (Bry and Schaffert 2002c). Xcerpt is "strongly answer-closed", i.e. it not only allows one to construct answers in the same data formats as the data queries like, e.g., XQuery, but also allows further processing of the data generated by this same query program. One of the design principles is to strictly separate the matching part and the construction part in a query. Xcerpt follows a pattern-based approach to querying the XML data. However, Xcerpt has extended the pattern-based approach in following aspects. Firstly, the query patterns can be specified by incomplete query

specifications in three dimensions. Incomplete query specifications allow the pattern specifications to be specified in a more flexible manner but without losing out the accuracy. Secondly, the simulation unification computes answer substitutions for the variables in the query pattern against underlying XML.

Unlike the popular XML query languages XQuery and XSLT using XPath expressions for selecting XML data, Xcerpt is a pattern-based language that avails of pattern matching algorithm called "simulation unification" (Bry and Schaffert 2002b) to select data from the XML data sources. Xcerpt uses rooted graph simulation to represent the query patterns and the XML data. XML data does not have a data structure as rigid as the relational data in the databases. For example, optional attributes in a composite XML Type may not appear in instances of the composite XML type. Therefore, Xcerpt supports incomplete query specifications in query terms. The unification process of query terms with incomplete query specification is different to the ones with complete query specification. For the former, unification is found if the graph induced by the direct children terms of the query term matches the graphs induced by the direct children terms of the data term. In contrast, for the latter, unification is deemed found if the graph induced by all children terms of the query in any depth and breadth matches the one induced by all children terms of the data term in any depth and breadth. In order to find the similarity between two graphs for incomplete query specifications, a non-standard unification called "simulate unification" is used to determine if the query term matches a data term in the case of the incomplete query specification. Therefore, it is particularly beneficial to the Web services-based architecture because the XML data coming from various Web service providers does not always conform to the global XML schema thoroughly with the level of flexibility in the XML schema definition.

However, Xcerpt is not supported natively by Oracle BPEL process manager that is our chosen runtime environment. The prime design target of Xcerpt is to query the Web and the Semantic Web; all the use cases given by the authors are limited to querying the XML documents from file systems. Therefore, extensions or workarounds are required for the architecture integration. For example, resources identifiers that are tied to an individual query make it difficult to construct query programs dynamically. The resources are specified as URLs in Xcerpt. However, in

the context of Web services architecture, the XML data are pushed into the transformation engine rather than being pulled by the transformation engine.

### 3.4 Summary

In this chapter, we have looked at approaches of XML data transformation with the intension of achieving a better modifiability of schema mapping rules. We also have discussed approaches that automate the process of generating schema mappings. We have drawn the conclusion that the best solution to address XML transformation is to use a XML transformation language with which integration rules can be expressed.

In the next chapter, we discuss the XML query and transformation. Many XML query and transformation language have been proposed to query the web, Semantic web, or the XML databases. In order to select the right language to best suit our needs, we outline requirements that have identified in the literature in the data integration domain and evaluate them in systematic fashion.

# Chapter 4
# Selection of a XML query and transformation language

In this chapter, we document the selection process of the most suitable XML query and transformation language for the given problem context in a systematic manner. The selected language will be integrated into the overall architecture to address data integration and mediation problems in the ASP problem domain. In Section 4.1, we motivate the selection process by outlining the problems in the XML Web service-based data integration systems. In order to choose the right language, we need to understand the principles of XML query and transformation languages, and we need to understand how a new XML query and transformation language is designed and motivated in the literature. Thus, in Section 4.2, we outline the principles of XML query and transformation, and the common practice in the literature in terms of the selection process. As a result, we define a new set of criteria by eliciting requirements from the problems for the selection process in Section 4.3. In Section 4.4, we select a few candidates from existing languages for comparison in more detail with examples and diagrams in order to find the right one to suit our settings. Finally, in Section 4.5, we evaluate each of these selected candidates with our new set of criteria to conclude that which language suit our business needs. We also look at the desirability of developing a new XML query language if no suitable language is found.

## 4.1 XML query and transformation problems

Domain experts who are responsible for identifying and verifying the correspondence do not have the programming experience. In practice, the first draft of the correspondence rules may be in some informal annotations such as being recorded into Excel spreadsheets or Word documents. Afterwards, developers start to express them with a XML query language once the correspondence rules are elicited. The rules expressed with a XML query language need to be verified by the domain experts again in order to make sure that the rules are expressed according to the original formalism. We summarize the XML query and transformation problems in our problem context in the following bullet points.

- **The language usability**. Correspondences rules are identified by the semantics of the schema definitions, the business rules, and the business documents. The most common operations on these rules are grouping elements and restructuring elements.
- **The evolution of correspondence rules.** Quite often, the construction of multiple objects in the global schema shares correspondence rules. As a result, elements in the global schema might have multiple parents. For example, A XML element at a lower level has parental nodes at higher-level branches of a XML tree. The query result has to be consistent to ensure the integrity of the global XML schema.
- **Views on integration rules.** During the development and maintenance stage of a data integration project, it is inevitable to alter querying rules because of fixing bugs or changes to the business requirements. Therefore, all these queries need to be modified accordingly. Furthermore, XML schemas evolve incrementally. In particular, the evolution to the XML schema and schema mappings has to be preserved and versioned.

## 4.2 XML data query and transformation

A XML query and transformation language is used to compose the transformation rules between the integrated schema and source schemas in such way that the composition of these queries can be constructed dynamically. Therefore, the rigid selection of an appropriate XML query and transformation language is crucial to the success of a data integration system. In Section 4.2.1, we look at the principles of some of XML query and transformation languages. Finally, we outline the desiderata for an XML query and transformation language with the intention to generate the transformation programs automatically.

### 4.2.1 Principles of XML query and transformation

XML query and transformation needs stronger expressive power than traditional database languages such as the relational algebra or SQL. XML data might not always have a schema that defines the structures and relationships of elements, and the XML data may contain nested structures. Characteristics of an XML query language have been studied extensively in the past (Maier 1998, Deutsch et al. 1999). However, they mainly focus on the features on how to query a XML

repository or XML documents in the spirit of database query languages rather than constructing a new XML document in the context of the data integration. The following principles for a XML query and language are inspired by the literature but do not necessarily correspond directly to them.

**The language should allow programmers to concentrate on the integration logic.** The language should be declarative enough to query and transform XML data at a higher abstract level.

**The Language should support both querying and restructuring XML data.** Firstly, the language should be able to answer queries imposed on the global schema and integrating a multitude of XML data sources. Secondly, the language should be able to answer queries by transforming the current data structure into a new structure according to the global schema.

**The language should support the generation of query programs.** The software technology enables queries to be generated automatically. For example, some XML query languages provide graphic user interfaces to enable XML queries generated automatically corresponding to the query building through GUI.

**The language should have strong expressive power.** In data integration systems, a XML query and transformation language queries XML documents, merges nodes from the various XML graphs that match the conditions specified in the transformation specification, and restructures selected nodes for the integrated schema. Furthermore, a single query should be capable of carrying out these operations rather than multiple XML queries to perform these operations (Deutsch et al. 1999).

**The language should be compositional.** The integration rules are derivation rules in data integration systems. Integration rules normally work together to transform XML documents conforming to the global XML schema. Therefore, it is essential for a XML query and transformation language to support query composition. Furthermore, any changes to the correspondences between elements and the XML schema definitions will not affect the query programs as a whole. On the other hand, using query composition can decompose large programs into the finer grained level in terms of modifiability. Additionally, using query composition, integration queries can be searchable in the rule repository, and the queries may be reusable to express

other queries. In order to make the queries that are capable of taking part in the composition, the querying part is better off being separated from construction part. In other words, no nested queries should be embedded in the construct pattern.

**The language should be rule-based.** The rule-based query languages are declarative. The variable bindings generated by specifying matching patterns in the rule body can be used in the rule head to construct the output. Rule-based languages may also use the derived facts in the matching part of other rules. The query program expressed in rule-based XML query languages consists of a set of rules, which composed together to answer users' queries. Being able to process XML documents, as a built-in functionality, is one of major design goals of some recently proposed functional programming languages. To name a few: XDuce (Hosoya and Buneman 2003), Cduce (Benzaken, Castagna, and Frisch 2003.), and the library HaXml (Wallace and Runciman 1999) adding XML processing support to the functional language Haskell.

**The integration suitability**. If the application context in this thesis is enterprise data integration, the runtime of a data integration language should be easily integrated into the overall software architecture. For example, some leading industrial products, such as Oracle BPEL process manager, have a built-in XSLT runtime engine so that any XSLT scripts can be easily integrated into as a part of the process. The suitability for integration also requires the runtime engine should be able to accept and process XML data coming from the hosting container and return the transformed or integrated data back to the hosting container.

### 4.2.2 Desiderata in the literature

Most XML and semistructured data query languages have been proposed to extract XML data from the XML databases or the web such as XPath, XSLT, XQuery, XML-QL. A comparative analysis of the existing languages has been done in the past (Bonifati and Ceri 2000). Some language specifies in database querying, others specify in data mediation for XML and semistructured data. In (Abiteboul, Buneman, and Suciu 2000), authors argue that one query language should be able to query data source using complex predicates, joins and even document restructuring. Without violation of our principles listed in Section 4.2.1, we set forth the following

criteria are compiled together based on the literature specifically in the context of data integration.

**The language should support joint of multiple XML data sources.** In data integration systems, the XML data comes from various data sources.

**Query results.** The output of the query should be XML rather than references or other functional language code.

**Incomplete queries.** XML and semistructured data are not as rigid as relational data in term of schema definitions and data structure. When querying the XML data, since the querying specification are built on the rules elicited from the XML schema definitions, the exact structure of the XML data might be unclear to the query language. Query incompleteness is also called partially specified path expression (Bonifati and Ceri 2000).

**Halt on cyclic query terms.** If a language supports querying represented with incomplete query specifications by wildcard and regular expression, it might cause query termination problem.

**Building new elements.** The ability to construct a new element or a new node adding to the answering tree is important feature in the data integration systems.

**Grouping.** Grouping XML nodes together by some conditions by querying the distinct values is very powerful feature in data integration also. Some languages use nested queries to perform grouping operation.

**Nested queries.** Nested queries are common in relational database languages in order to join data elements from various resources by their values. In logic based languages, the construct part and the selection part are separated.

**Tag variables.** Label variables are introduced to address scenarios where labels need to be specified in a query rather than the data. Unlike a conventional query language, we do not need to know the value of a query term in order to transform a query term. In some data integration systems, the schema for a data source is unknown beforehand; therefore, a label variable is important to retrieve structural information.

**Query reduction.** Query reduction allows users to specify what part of the elements or what nodes in the querying conditions to be removed from the resulting XML tree.

### 4.3 New desiderata for a XML query and transformation language

We have outlined the XML query and transformation problems in Web service-based data integration architectures in Section 4.1, also studied the principles and the selection process of a language in the literature in Section 4.2. In this section, we discuss the new desiderata we have come up from the problem setting and problems specific requirements, which is to generate the query programs in an automatic manner. The new desiderata are added onto the list identified in Section 4.2.2.

**Code reuse**. Component-based software engineering helps code reuse because of clear definition of interfaces and boundaries among components. Therefore, we argue that the modular programming model is essential for a XML query and transformation language.

**Easy of use.** Three most desired qualities for an XML query and transformation language is declarative, expressive power and ease of use (Bonifati and Ceri 2000). Data integration rules are valuable assets to an organization, before they are implemented in a data integration project, queries expressing integration rules have to be verified and checked up by domain experts. Therefore, the readability of the queries will be another desired feature.

**Pattern queries**. In order to query and transform the XML data, the means for accessing or selecting data have to be defined. The means of accessing data should be able to reach to arbitrary depths in the XML data graph. Most query and transformation languages for XML specify the structure of the XML data to retrieve using one of the following two approaches. One is navigational approach; the other is pattern-based approach. In the former approach, the data can be accessed via the notion of a path expression. One of the most distinguished features of this approach is that there is only one path expression that is evaluated per query. Current XML query languages like XQuery or XSLT use a navigational approach to select data items in such tree structures. In the latter approach, augmenting the syntax for query expressions with variables specifies matching patterns on the XML structure to express complex conditions. Query patterns contain variable placeholders for the

latter binding to nodes during the evaluation process. A pattern specifies a data graph that may be matched to some larger graph. Patterns are also useful form the multiple bindings that are needed to express joints. There is no explicit select (i.e. no navigation through a hierarchy) to select the nodes. In a positional or pattern-based approach, a query pattern is like a form that gives an example of the data for selection in QBE (Zoof 1977). The positional languages use expressions that mimic the data in the query terms.

**Strict separation of construction and matching conditions**. Some of XML query languages have to mix them in a nested way so that the generation query programs are difficult and it is difficult to read also. Unlike queries on traditional relational databases whose results are always flat relations; the results for XML queries are complex. Thus, XML queries are better off strictly separating two components: querying part and result constructing part in a query specification.

**Answers as queries**. In the pattern based query language, query patterns are like the form filling with data. The variables are bind to the form. If the answer to a sub-query is replaceable by an answer, the query is still valid. This requirement is essential to support rule chaining and query composition. Wherever a variable denoting a query term is used in a query program, the variable should be replaceable with another expression (Abiteboul, Buneman, and Suciu 2000).

## 4.4 A comparison of XML query and transformation languages

In this section, we introduce a list of candidates for the selection process. There are two specification approaches to specify data selection: one is navigation approach; the other is pattern-based approach. Therefore, we select a few from each side of the data selection. XPath is the typical example of navigation approach and integrated into XSLT and XQuery for data selection. XSL/XSLT is currently the most frequently used transformation language in Web service-based data integration architectures and commercial tools. XQuery is a W3C standard language as a potential replacement for XSLT. It also has been used in a XML database management system. XML-QL is a declarative rule-based language with the intension of querying and transforming XML data only. XML-QL uses patterns matching to select data. Xcerpt also follows a pattern-based approach to querying the

XML data. However, Xcerpt has extended the pattern-based approach by so-called "*simulate unification*" (Bry and Schaffert 2002c).

For each of these languages, we briefly introduce the overview of the language, and then we evaluate how the language works with examples in terms of the criteria identified in Section 4.2 and Section 4.3.

### 4.4.1 Use case

Examples illustrate the syntax and the functionality of the candidates based on the following use case in this section throughout this chapter. An enterprise "ERP Online" manages other companies' applications running environment including both the software and hardware. When a customer comes on board, it will be assigned a "`Customer service identifier`" and a "`Customer identifier`". To illustrate how "`APS`" manage its customers, we imagine a customer named "`Buy & Sale Inc.`". ERP Online is a large global company with offices around the world. On March 4 2003, the IT department of Buy & Sale Inc. signed a contract for 30 users of ERP application license from the ERP Online. As agreed in the contract, ERP Online supplies and maintains hardware and software in-house. On August 4 2005, further 40 users license for ERP Online's CRM application, and a 100 user's license for the HR and Financial products. Buy & Sale Online's contracts with ERP Online resulted in two major implementations. The first implementation was the provision of a global single instance of ERP application in Buy & Sale Online in Ireland and the US. ERP Online in China required a separate instance of ERP application to handle double bytes Chinese character sets.

The XML document "`customer.xml`" in Fig. 4-3 contains the data of such a customer manager system. The customer management database uses the following schema definition in Fig. 4-1). The Customer identifier and the customer name represent a customer; each customer might have multiple customer service identifiers. A customer might have multiple operations globally. A Country code and a customer service identifier identify each of the customer's operation.

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
           <xs:element name="arrayOfCustomer"
           type="arrayOfCustomerType"/>
     <xs:complexType name="arrayOfCustomerType">
                      <xs:sequence>
```

```
                    <xs:element name="customer"
                    type="itemType"
                    maxOccurs="unbounded"/>
               </xs:sequence>
          </xs:complexType>
          <xs:complexType name="itemType">
               <xs:sequence>
               <xs:element name="orgName"
          type="xs:string"/>
                 <xs:element name="companyId"
          type="xs:string"/>
                    <xs:element name="countryCode"
                    type="xs:string"/>
                 <xs:element name="csiNumber"
          type="xs:string"/>
               </xs:sequence>
          </xs:complexType>
</xs:schema>
```

Fig. 4-1 The XML schema for the customer management Web service provider.

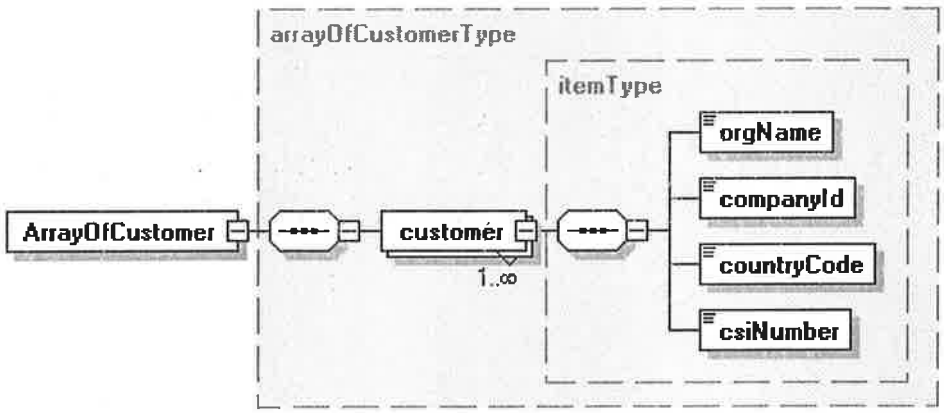


Fig. 4-2 The diagram of XML schema for the customer object.

```
<arrayOfCustomer>
     <customer>
          <orgName>Buy and Sale Online</orgName>
          <companyId>90</companyId>
          <countryCode>840</countryCode>
          <csiNumber>1398</csiNumber>
     </customer>
     <customer>
          <orgName>Buy and Sale Online</orgName>
          <companyId>90</companyId>
          <countryCode>372</countryCode>
          <csiNumber>1399</csiNumber>
     </customer>
     <customer>
          <orgName>B2BOnline.com</orgName>
            <companyId>99</companyId>
          <countryCode>372</countryCode>
          <csiNumber>1400</csiNumber>
```

```
        </customer>
    </arrayOfCustomer>
```

Fig. 4-3 An instance of the XML schema definition for the customer object.

### 4.4.2 XPath

XPath provides expressions for selecting data by means of navigating through the graph representation of an XML document. XPath cannot be considered a true XML query and transformation language because it works as a data selection language for other languages such as XQuery and XSLT.

**Path expressions**. The core expressions of XPath are "location steps". A location step specifies where to navigate from the so-called "context node", i.e., the current node of a path traversal. In order to select the "`companyId`" for a given customer name "`B2Bonline.com`". An XPath expression starts the matching process from the document node. Proceed to the element node customer from the left to the right. For each node "`customer`", each child of the node will be accessed to determine whether the name is "B2Boline.com". In this case, it selects in the next step that child node with label "`99`":

```
/child::ArrayOfCustomer/child::customer[
      child::customerNmae="B2Bonline.com"
]/companyId
```

Fig. 4-4 A XPath example.

A location step consists of three parts: an axis, a node-test, and an optional predicate. The axis specifies candidate nodes in terms of the tree data model. The base axes self, child, following-sibling, and following (selecting the context node, their children, their siblings, or all elements if they occur later in document order). The transitive and transitive reflexive closure axes descendant and descendant-or-self of the axis child, and the respective reverse (or inverse) axes parent, preceding-sibling, preceding, ancestor, and ancestor-or-self. Two additional axes, attributes and namespace, give access to attributes and namespace declarations. Both node-tests and predicates serve to restrict the set of candidate nodes selected by an axis.

XPath has no means for defining variables, as it is embedded in a host language such as XSLT or XQuery that may provide such means.

### 4.4.3 XSL/XSLT

XSLT the Extensible Style sheet Language is a language for transforming XML documents. It has been adopted quickly because it was the first W3C language for XML query and transformation. Transformation is here understood as the process of creating a new XML document based upon a given one. An XSLT program is composed of one or more transformation rules (called templates) that recursively operate on a single input document. Each rule consists of a pattern and a template. Transformation rules are expressed with XPath expressions. XSLT starts from the root element and tries to apply a pattern to that node.

**Template rules and named templates.** A template can consist of the resulting elements matched by the guard expression and a selection of elements in the input tree. The selection of the elements to process further is done using an XPath expression. If no specific restriction is given, all templates with guards matching these elements are considered, but one can also specify a single (named) template or a group of templates. The XSLT stylesheet in Fig. 4-5 creates another XML document by renaming the tag "orgName" to "customerName" and "arrayOfCustomer" to "CustomerList".

```
<xsl:template match="arrayOfCustomer">
      <CustomerArray>
            <xsl:apply-templates select="./customer"/>
      </CustomerArray>
</xsl:template>
<xsl:template match="customer">
      <customer>
      <xsl:if test="customer/orgName[@xsi:nil='true'])"
      <xsl:element name="customerName">
            <xsl:value-of select="./orgName"/>
      </xsl:element>
      </xsl:if>
      <xsl:if test="customer/companyId[@xsi:nil='true'])"
      <xsl:element name="companyId">
            <xsl:value-of select="./companyId"/>
      </xsl:element>
      </xsl:if>
      <xsl:element name="countryCode">
            <xsl:value-of select="./countryCode"/>
      </xsl:element>
      <xsl:element name="csiNumber">
            <xsl:value-of select="./csiNumber "/>
      </xsl:element>
      </customer>
</xsl:template>
```

Fig. 4-5 XSLT stylesheet template rules.

**Structural Recursion.** It starts from the root "<arrayOfCustomer>...</arrayOfCustomer>" and tries to match some pattern to the root node. Therefore, XSL evaluates the template body, and this determines the whole program to be applied to all the children. The ability of traversal of data followed by the reconstruction of a new XML document is called structural recursion. It is the fundamental computation model of XSLT. The recursion starts from the root element, the imaginary root and traverses to the leaves. The template rules are applied while their patterns match and the result selected by the XPath expressions is written to the result document.

**Restructuring queries.** XSLT provides a set of programming constructs that are capable of constructing new elements and grouping elements. The construct "xsl:for-each" iterates over all elements for a set of nodes selected by an XPath expression. The "xsl:if " construct generates certain part of nodes if the matching condition is met. The "xsl:choose" construct allows specifying several alternative options guided by conditions.

**Incomplete query specification.** XSLT support both relative and absolute locations.

**Query reduction.** XSLT does not support query reduction directly, but users are able to specify templates rules and matching patterns to realize query reductions in Fig. 4-6:

```
<xsl:template match="@*|node()">
      <xsl:copy>
            <xsl:apply-templates
      select="@*|node()"/>
      </xsl:copy>
  </xsl:template>
  <xsl:template match="csiNumber | countryCode" />
```

Fig. 4-6 The reductive query in XSLT.

**Grouping.** XSLT does not support grouping directly. In order to perform grouping operation in XSLT, users need imperative programming language to do it as in java or C. To illustrate how to do grouping and constructing new elements in XSLT, we take the following example to demonstrate it. We group the customers together by the distinct "companyId" and introduce a new element called "supportIdentifier" that is made up of "csi_number" and "country_code".

The example in Fig. 4-7 generates a customer with multiple "supportIdentifiers".

```
<xsl:template match="arrayOfCustomer">
      <CustomerArray>
      <xsl:for-each select="customer">
      <xsl:sort select="gcdbOrgId" />
            <xsl:variable name=" thisCompanyId"
            select="companyId" />
            <xsl:if test="not(preceding-
            sibling::item[companyId=$thisCompanyId])">
                  <Customer>
                        <xsl:element name="customerName">
                              <xsl:value-of
                        select="orgName"/>
                        </xsl:element>
                        <xsl:element name="companyID">
                              <xsl:value-of
                        select="companyId"/>
                        </xsl:element>
                        <xsl:for-each
                        select="../customer[thisCompanyId=
                        $ompanyId]">
                        <xsl:element
                        name="supportIdentifier">
                        <xsl:element
                        name="CustomerSupportIdentifier">
                              <xsl:value-of
                        select="csiNumber"/>
                        </xsl:element>
                        <xsl:element
                  name="ISOCountryCode">
                              <xsl:value-of
                              select="countryCode"/>
                        </xsl:element>
                        </xsl:for-each>
                  </Customer>
            </xsl:if>
      </xsl:sort>
      </xsl:for-each>
      </CustomerArray>
</xsl:template>
```

Fig. 4-7 Grouping and constructing new elements in XSLT.

### 4.4.4 XQuery

A few other languages such as XPath, XSLT and XML-QL influence the design and development of XQuery by the XML Query Working Group. The design principles are given in (Katz 2004). For example, it uses the similar Path expression as XPath version 1.0, the XPath 2.0 is a part of XQuery 1.0; it leverages the typing systems from the XML and the XML schema. The expressions that can be composed further

consist of document constructors, element constructors and iterative expressions. From XML-QL, XQuery learned the power of binding variables to sequences of values so that the new elements can be constructed on top of the bound variables. We start to evaluate the language based on the following criteria in bold. The book XQuery from the Experts (Katz 2004) inspires the following evaluation.

**Incomplete query specification.** XPath plays a key role in XQuery in terms of selecting nodes in the tree structure. Therefore, it supports incomplete query specification just in the same way as in XPath.

**Join.** XQuery is designed with the intension of querying database as well as the documents on the web. Therefore, not only multiple variables does it support in the loop, also joining them together based on the matching conditions expressed in the WHERE clause.

**Constructing and restructuring queries.** Element constructor is one of the strength of XQuery in terms of constructing new XML element. The Element constructor follows closely to the XML notation. XQuery also provides a way to construct a new node by using the computed names as well as other kinds of nodes. For example, elements such as "{$name}" and "{$content}" constructs an element node. The variables can be replaced by XQuery expression.

**FLWOR expressions.** An expression of the form "$name" is called a variable reference. Variables may be bound to values, i.e. sequences of sub trees selected by XPath expressions. Variable references may be used in XPath expressions where they are substituted by their binding. If the value of the binding contains more than one item, then each of these items is substituted in turn, building a union of all selected data items **"FOR"** and **"LET"** serve to bind variables in various manners. **"FOR"** iterates over all items in a sequence and binds a variable successively to each item, **"LET"** binds the variable once to the complete sequence. We illustrate the usage of FLWOR expressions with the use case in Section 4.4.1. The query can be expressed in XQuery as follows to select all the customers with the country code are "372" in Fig. 4-8.

```
<arrayOfCustomer> {
      let $customers :=
doc("file:customer.xml")/arrayOfCustomer
      for $customer in $customers/customer
      order by $customer
```

```
        where $customer/countryCode = 372
        return
        <customer>
              {$customer}
        </customer>
</arrayOfCustomer>}
```

Fig. 4-8 The construction with FLWOR expression in XQuery.

**Grouping**. FLWOR expressions can also be nested to express grouping construct. For example, for each customer in the customer management database, group the customers by their "countryCode". This example also inverts the relation between a customer and a country in Fig. 4-9.

```
<customerArray>
  {
      let $customer :=
      doc("file:customer.xml")//customer
      for $country in distinct-
      values($customer/countryCode)
      order by $country
      return
         <countryCode>
            {
                  for $c in
                  doc("file:customer.xml")/customer
                  where some $ca in $c/countryCode
                        satisfies ($ca = $country)
                  return
                  { $c }
             }
          </countryCode>
  }
</customerArray>
```

Fig. 4-9 The grouping representation in XQuery.

This implementation is quite similar to the implementation of use case XMP-Q4 in (Chamberlin et al. 2005). There are two unique functionalities in this example: The use of distinct-values to avoid data duplication. The use of existential quantifier some to find customers have the same "countryCode" as the currently consider customer.

### 4.4.5 XML-QL

XML-QL is a rule-based query language for XML developed specifically to address the W3C's call for an XML query language that resulted in the development of XQuery. XML-QL uses query patterns and path expressions to select data from the XML sources. One of the main characteristics of XML-QL is that it uses query

patterns containing multiple variables that may select several data items at a time instead of path selections that selects one data item at a time. The variables in XML-QL are similar to the variables of logic programming. However, XML-QL uses nested queries to perform complex queries and grouping the XML data. XML-QL is designed specially to query the XML data in data integration applications.

**Incomplete query specification.** Since XML data might have optional elements, XML-QL provides a flexible query expression to express the optional elements. For example, the "<countryCode>" tag in "customer.xml" might be optional. The following query is to select all the "csiNumber" and where available, the "countryCode" also. XML-QL uses nested queries to handle optional parts n in Fig. 4-10.

```
where customer> $C </customer> in "file:customer.xml",
                <csiNumber> $CSI </csiNumber>
                in $C
                construct
                      <result>
                            <csiNumber> $CSI </csiNumber>
                where <countryCode> $CC </countryCode>
                in $C
                construct
                      <countryCode> $CC </countryCode>
                </result>
```

Fig. 4-10 The incomplete query in XML-QL.

**Restructuring queries.** Consider the example given in Fig. 4-11 retrieving all the "csi" numbers of customers "B2BOnline.com".

```
WHERE
      <arrayOfCustomer>
            <customer>
                  <customerName>B2BOnline.com</custo
            merName>
                  <companyId> $ID </companyId>
                  <csiNumber> $CSI </csiNumber>
                  <countryCode> $CC </countryCode>
            </customer>
        IN "file:customer.xml"
CONSTRUCT $CSI
```

Fig. 4-11 The reconstruction of the XML data in XML-QL.

In the above example, the content between keyword where and in is a pattern. The query processor will match the pattern in all possible ways to the data and bind the variables "$ID", "$CSI" and "$CC". For each binding it will produce a result "$CSI".

**Grouping with nested queries.** In XML-QL, grouping is expressed by nested queries.

### 4.4.6 Xcerpt

What make Xcerpt stand out from the other XML query languages are the pattern-based query specification and the data integration technique such as rule chaining and simulate unification. The following design principles guide the design of Xcerpt language. An elaborate discussion on the design principle of Xcerpt can be found in (Bry and Schaffert 2002a).

**Declarative rule-based query language**. An Xcerpt query program consists of some rules expressed in form of construct...from...where...end and a goal expression. Xcerpt is a compact, high-level language that has an expressive power strong enough to express complex query programming by ways of rule chaining. Xcerpt encapsulates the completeness of grouping elements by integrating grouping data as a built-in feature.

**Pattern-based queries**. Xcerpt uses patterns for both selecting data and constructing and transforming the selected data as apposed to the most of existing XML query languages that use paths for selecting data.

**Separation of querying and construction**. Xcerpt follows strict separation of query terms and construct terms. A query term dedicates to retrieving data by generating appropriate substitutions generated by the variable bindings between query terms and data terms. A construct term dedicates to restructuring the data structure or creating the new data by applying the generated substitutions.

**Backward rule chaining**. Backward rule chaining is a goal driven approach. In Xcerpt, the result of a query can be queried by other queries. For example, a composite query may consist of nested queries that are represented by variables. In data integration applications, it is important for the composite objects that are able to be composed together by sub queries to realize views.

**Incomplete query patterns**. Incomplete data structure such as difference-list is another one of the established technique. XML data can be treated as semistructured data in terms of querying operations. Query pattern incompleteness provides very strong querying mechanism to construct new data and transform the source data to

facilitate the productivity. In order to realize query pattern incompleteness, Xcerpt introduces a novel approach to unify queries instead of strict unification as in logic programming languages called "simulation unification". The heart of the computation model of logic programs is unification. A unifier of two query expressions is a substitution making two query expressions identical. If two query expressions have a unifier, we call say they unify. All the substitutions are expressed by variables.

Construct terms construct new data terms by combining variables defined in the query terms and the construct patterns. The construct patterns can be viewed as the pre-defined template and the structure of the data terms in which the variables can be filled in. The outcome of the simulation unification is a set of substitutions for the variables in the query term. A substitution is a mapping from all the variables to the construct terms that cannot be applied to the construct terms directly because of grouping constructs "`all`" and "`some`". Therefore, a substitution set is introduced to apply onto the construct terms. The construct term given in Fig. 4-12 creates a new element "`supportIdentifier`" that consists of the element "`csiNumber`" and the element "`companyId`". The variables "`csiNumber`" and "`companyId`" are expressed in the query term. Note: the incomplete specification cannot be used in the construct terms.

```
SuppoerIdentifier [
      var csiNumber,
      var companyId
]
```

Fig. 4-12 The construct term in Xcerpt.

**Group construct "all"**. The XML data produced by web services, exposed by the data sources which is normally a relational database, have a flat data structure, one of major data transformation tasks is to transformation the flat XML data into a more object oriented data model. The "all" construct groups all matching instances to a variable corresponding to the enclosed children terms in a query term and resembles the sequences in an answer. Nested grouping constructs are used to perform more complex transformation tasks. The following example in Fig. 4-13 illustrates how Xcerpt handles grouping.

```
GOAL
```

```
out {
      resource { "file:customer.xml", "xml" },
      CustomerArray [
            all   customer[
                  var name,
                  companyId[var CompanyId],
                  all supportidentifier[
                        CustomerSupportIdentifier[var
                  Code],
                        ISOCountryCode [var CSI]
                  ]
            ]
      ]
}
FROM
      IN {
      resource { "file:customer.xml", "xml" },
      arrayOfCustomer[[
            customer [[
                  var name -> customerName,
                  companyId[var CompanyId],
                  countryCode[var Code],
                  csiNumber[var CSI]
            ]]
      ]]
}
END
```

Fig. 4-13 The reconstruction of the XML data in Xcerpt.

Xcerpt allows to group answers using the constructs "`all`" and "`some`". "`all`" renders all possible instances of the substitutions that might result from multiple variable bindings. In the example in Fig. 4-13, the "`all`" construct might also be nested, the construct term creates a customer term for each alternative bindings of customer, and within each resulting customer child term, it groups all "`cisNumbers`" and "`countryCodes`" associated with that particular customer. The "[[" symbol denotes the incomplete specification of query pattern. For example, if the element "`customerName`" is optional, the customers without a "`customerName` " element can still be binding to the variable during the query processing.

## 4.5 Evaluation

In this section, we summarize the evaluation based on these criteria given in Table 4-1, and then briefly discuss each of the five languages accordingly. We also discuss the output of the evaluation after the selection process. Finally, we discuss the

feasibility of integrating the chosen language into the Web service-based data integration architecture.

Table 4-1 Comparison of the five XML query languages.

| **Criteria** | **XML-QL** | **XSLT** | **XQuery** | **Xcerpt** |
|---|---|---|---|---|
| Joins | No | No | Yes | Yes |
| Query Results | XML | XML | XML | XML |
| Incomplete Query Specification | Yes | Yes | Yes | Yes |
| Construct new elements | Yes | Yes | Yes | Yes |
| Tag variables | Yes | No | No | Yes |
| Halt on cyclic query terms | N/A | N/A | N/A | Yes |
| Nested queries | Yes | Yes | Yes | Yes |
| Grouping | Yes | No | Yes | Yes |
| Easy to use | Yes | No | No | Yes |
| Query/construct separation | No | No | No | Yes |
| Answer as query | No | No | No | Yes |
| Pattern-based queries | Partly | No | No | Yes |
| Query Reduction | No | No | No | Yes |
| Modifiability | Low | High | High | Low |
| Tool Support | Little | Sufficient | Sufficient | Prototype Implementation |
| Application Domain | Web Application Data Integration | Generic transformation | Data Integration Or The Web | The Web or Semantic Web |
| The integration Suitability | Not supported by Oracle | Built-in runtime support | Not supported by Oracle | Not supported by Oracle |

| | | | | |
|---|---|---|---|---|
| | BPEL | | BPEL | BPEL |

XSL/XSLT is a style sheet language that generates another XML document by transforming and querying a single XML document, lacking joins and grouping construct have made it as the first language to be out of competition.

XML-QL has powerful expressive power such as incomplete query specification, constructing new elements and supporting tag variables. XML-QL is also a pattern-based query language that resembles queries as filling a form like process. However, XML-QL does not have grouping construct. Nesting queries together mixes the query pattern and the construction pattern.

XQuery is a W3C standard language that has scored well in our evaluation. However, the navigational path expression adopted from XPath has brought some downsides. The query result of XQuery cannot be used directly as a valid query. The XQuery uses sub-queries for grouping that makes it impossible to separate query pattern and construct pattern strictly.

Xcerpt scores very well in the overall evaluation. It has the functionality we need for our architecture in terms of XML query and transformation although it has only prototype implementation. Many research activities including applications and even extensions have been proposed or conducted (Berger et al. 2005, Bolzer 2005).

Another possibility is that we develop our own tailored language especially for the Web service-based integration architecture. However, there are already numerous languages there for XML query and transformation although prime application domain of which is for SOA. We have ruled out this option because designing a new language is a complex and lengthy task and we try not to reinvent the wheel.

Therefore, our final decision out of the selection process is to use Xcerpt for our solution towards XML Web service-based data integration architecture. The comparison conducted in Bonifati and Ceri (2000) has inspired this selection process and criteria used in this chapter. In particular, we added XQuery and Xcerpt to the candidates list, some criteria have been tailored up and several new criteria based on our application context have been added into the list.

## 4.6 Summary

In this chapter, we discussed the design principles of XML query and transformation languages in general, we discussed the XML query and transformation problems that we faced in detail. Thus, we have elicited more criteria in order to choose the most suitable one from many candidate languages inspired by the criteria set in the literature. Finally, we have drawn a conclusion that Xcerpt will be the language to be integrated into our data integration architecture after a systematic evaluation based on the criteria.

In Chapter 5, we describe our proposed architecture in detail, and observe the possibility of integrating Xcerpt into the data integration SOA and we find that extensions to Xcerpt are needed for integration. For example, how to pass the XML documents into the Xcerpt runtime environment without saving them into file systems at first, how to query the data integration rules in Xcerpt.

# Chapter 5
# Integration of Xcerpt into a mediated architecture

In this chapter, we integrate Xcerpt into the proposed software architecture to improve the modifiability of XML transformation. In Section 5.1, we introduce the system design of the proposed mediated software architecture, and how Xcerpt query programs can be used to improve the modifiability aspect of data integration. In Section 5.2, we discuss the integration related problems Xcerpt has and propose a solution to it for easy integration into the architecture. We also describe how the Xcerpt needs to be enhanced so that it can support the dynamic generation of Xcerpt query programs. In Section 5.3, we describe the proposed software architecture and its components in detail. In Section 5.4, we describe how a mediator Web service component is designed and constructed.

## 5.1 The system design of the mediated architecture

There are essentially three players in SOA: a service provider, a service broker, a service requestor and a UDDI Service directory. Despite the importance of having a UDDI directory specification, it has not been adopted widely as some expected. Many Web Service-based architectural designs have adopted a phase-based approach: the first step is to migrate to the current architecture to SOA within the organization. One of the key benefits that SOA brings to us is the flexibility of selection of service providers. However, in the application context of ASP setting, more than one service provider is not always an option. In fact, a service broker is in fact the mediated services or business processes integrator in data integration SOA hosted internal to the enterprise. In this thesis context, the service delivery environment in ASP as an organization might have only one data source for "Customer" data, one data sources for "Service request" data. Therefore, it is unrealistic to think that it might be an alternative to choose. Web service providers develop their Web services with requirements from the service integrator in mind or Web services are only developed for the internal users. UDDI is merely used internally during the development environment. Therefore, the UDDI directory is not as significant as the ones on the Web. Potential consumers can be huge even with an organization because each of the consumers tries to report the services for customers from another angle. The portal application in the thesis context is only one of them. In the following section, we

lists design principles that govern the design of our proposed software architecture. The two major contributions of the proposed mediated software architecture to the knowledge of the integration and mediation of XML data in the context of Web services are:

- The declarative rule-based query programs improves the modifiability;
- The automatic connector construction improves the reusability of the declarative integration rules.

### 5.1.1 Design principles

The aim of this thesis is to introduce a new technique to improve the modifiability aspect of the mediated Web services architecture for the integration and mediation of XML data. All design principles are in line with improving the modifiability.

**The component-based architecture**. The Conventional component-based middleware on the Web technologies platform have little success because it does not solve integration problems such as the differences in data models or business rules (Stal 2002). Zhu (Zhu et al. 2004) also argues that traditional data integration approaches such as federated schema systems and data warehouses fail to meet the requirements of constantly changing and adaptive environments. Traditional data integration solutions have tried to solve our research problems but its use of proprietary interfaces and communication protocols adds another layer of heterogeneity onto it. However, with the support of open-standards Web service technology, it is possible to encapsulate integration logic into a separate component to the mediator Web services and data Web service providers within the proposed software architecture. Therefore, we decide to build the connector construction component in Fig. 5-1 as a separate Web service component to minimize the interoperability with the mediator Web services.

The connector construction component is responsible for providing connectors to integrate and mediate XML documents. The mediator Web services component has an integration flow that is responsible for the integration and mediation of XML data. Between the mediator and various Web service providers, the mediator is responsible for invoking Web services providers and gathering XML documents from them. Between the mediator and the connector construction component, the

mediator is responsible for passing all interim XML documents into the latter and retrieving the mediated XML document back.

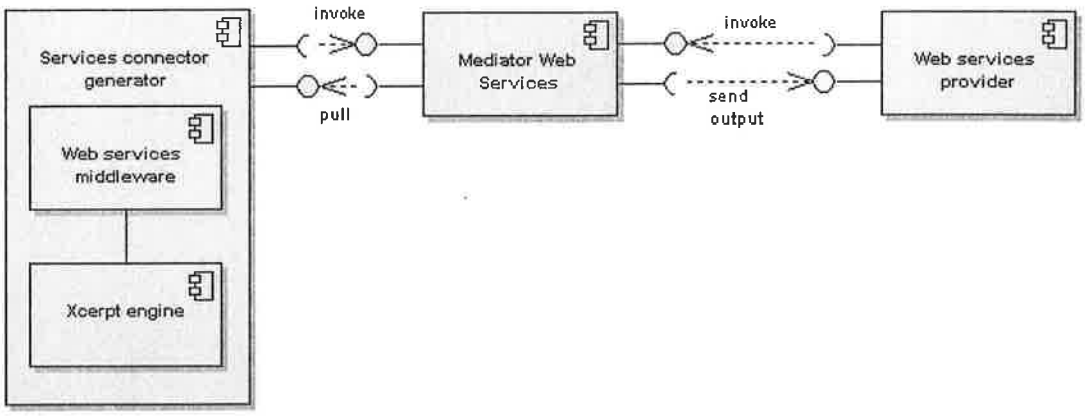


Fig. 5-1 The integration of the connector construction component.

The component-based development helps maintain the loosely coupled nature of SOA. All the interactions among components should be via published APIs such as interfaces in WSDL files. In order to build SOAP Web services, we need to wrap the SOAP/XML request in a HTTP request for outbound XML documents and to extract the SOAP/XML from the HTTP requests for inbound XML document.

**Xcerpt query programs should be constructed dynamically**. Producing appropriate communication glue code is mandatory before Web services providers and mediator web services can be connected to each other in order to inject no communication code into integration flows within mediator Web services. The connector component produces a connector based on the adaptive configuration such as the name of the mediator Web services, Web service providers. The separated connector construction components can be deployed as another Web services components in the architecture to connect Web service providers at runtime. As a result, less code is written for the mediators that serve as the service integrator role. Thus, the modifiability aspect of the architecture can be improved.

### 5.1.2 The component model of the connector construction component

One of approaches to make the Xcerpt runtime engine as a Web service component is to build a wrapper around it. An alternative approach is to develop Java APIs to interact with Xcerpt queries seamlessly like XQJ (Eisenberg and Melton 2004) to XQuery or JDBC to SQL. In order to promote the code reuse, an individual integration rule should not perform the transformation tasks alone rather than

working together in forms of composition. The composition of rules demands the query part of the connector built ahead of the construction part of the connector. The data presentation of the global data model changes such as changes to element names and removing elements should not affect the query and integration part of the logic. Only an additional construction part is required to enable multiple versioning of the global data model.

Connectors are constructed dynamically based on certain input data such as an array of source Web services and a mediator Web service. Ground rules are responsible for populating the XML data as the Xcerpt data terms by reading XML documents from individual Web service providers. These ground rules that are quite tightly coupled to individual Web services because the rules instruct the connector where to retrieve elements of data objects. The Xcerpt data terms are consumed subsequently by non-ground queries such as intermediate composite rules. The intermediate composite rules are responsible for integrating ground rules to render data types in the global XML schema. However, the intermediate composite rules still do not produce output from connectors for the mediator Web services. Finally, the composite rules are responsible for rendering the data objects defined in the interfaces of the mediator Web services based on the customers' requests, The composite rules are views on top of ground representations according to the global XML schema. Therefore, the exported data from a mediator Web service is the goal of the corresponding connector (a query program). The following section outlines how a composite rule is queried based on the customers' requests in Fig. 5-2

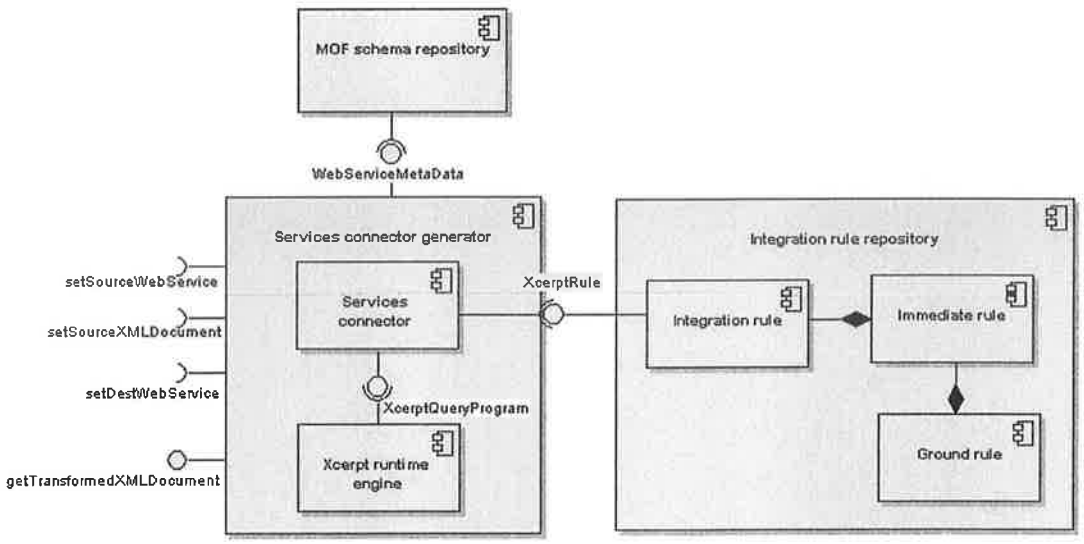


Fig. 5-2 The component model of the connector construction component.

### 5.1.3 Automatic the construction of connectors

The construction of a service connector in this context is to generate an Xcerpt query programs by compositing each of Xcerpt queries that correspond to integration rules. In an Xcerpt query program, there is only one goal query that will be processed at first by the Xcerpt runtime engine. The goal query is made up of composite queries that in turn are made up of ground queries that read XML data from external resources. We begin by expanding each composite query according to the definitional data mappings that are stored in the rule repository. The rule chaining mechanism in Xcerpt needs goal query and all the supporting queries in one query program at runtime. Therefore, we need to construct the query program in order to process incoming XML documents although the integration queries can be stored separately in the repository at deployment time.

Connectors can be used in the following two scenarios. Firstly, output XML document needs data mediation and transformation from the previous data Web service provider before it can be passed into next-in-line Web services in a WS-BPEL process flow. Secondly, output generated from the integration flow needs the connector to mediate data according to the global XML schema before the output is returned back. However, we think that the second scenario can demonstrate the declarative rule-based approach in a more meaningful manner because it is related to one of our motivating problems. Therefore, in this section, we use the second scenario to demonstrate the construction process.

- *Web Service Identifier: setSourceWebService (p_name: String, p_version: String)*

This interface is used to specify single or multiple sourced Web services from which the connector is generated. You will have to call it multiple times if the sourced Web services are more than one. Make sure the web service is registered into the MOF XML schema Repository in advance. The passing-in parameters are the name and the version of a web service. This interface returns the internal identifier returned from the MOF XML schema Repository.

- *SetSourceXMLDocument (p_webServiceIdentifier: number, p_xml: XMLDoc)*

This interface sets the XML document that needs transformation. You will have to call it multiple times if the sourced Web services are more than one. This interface

works in the pair with the interface "setSourceWebService()". One of the passing-in parameters is the Web service identifier where the XML document comes; the other is the source XML document.

- *WebServiceIdentifier: setDescWebService (p_name: String, p_version: String)*

This interface is used to specify for which mediator service the connector is constructed. The Web service has to be registered into the MOF XML schema Repository in advance. The parameters to the interface are he name and the version of a web service. Finally, the interface returns the internal identifier returned from the WS schema Repository.

- *XcerptConstruct: getXcerptConstructs (p_array_source_webservices Number_table, p_dest_webservice Number)*

This interface looks up the integration rules from the Transformation Rule Repository. The parameters to the interface are the following: the array of source Web services identifiers and the specified mediator web service identifier. The interface returns the Xcerpt constructs that enable the generated Xcerpt query program to reach the goal.

- *XcerptQueryProgram: ServicesConnector (p_source_xml_document_array, p_xcerpt_construct)*

At this point, all the prerequisite conditions have been met such as sourced XML documents and the metadata of sourced Web Services. Therefore, he Xcerpt query program will be able to generated and executed by the Xcerpt runtime engine. The parameters of this interface are the following: the array of source Web services identifiers and the specified mediator web service identifier. The interface returns a well-formed Xcerpt query program for execution in the Xcerpt runtime engine.

- *XmlDoc: getTransformatedXMLDoc (p_name: mediatorWebServiceIdentifier)*

This interface is called to return the transformed XML document back to the caller. The parameters passing into this interface are: The identifier of the mediator service specified in the interface "setDescWebService". The interface returns the transformed XML document that conforms to the global XML Schema.

We illustrate the basic flow of activity for processing input XML documents, the construction of a connector and the generation of the transformed XML document in

Fig. 5-3. We encapsulate all the data mediation related objects behind the Web service interfaces of the connector construction component. Passing-in parameters such as source Web services determine how the connector is constructed in the connector construction component. The construction of the connector is the process of realizing the query program with the goal rule. All the dependent integration rules of the goal will be picked up from the transformation rule repository.

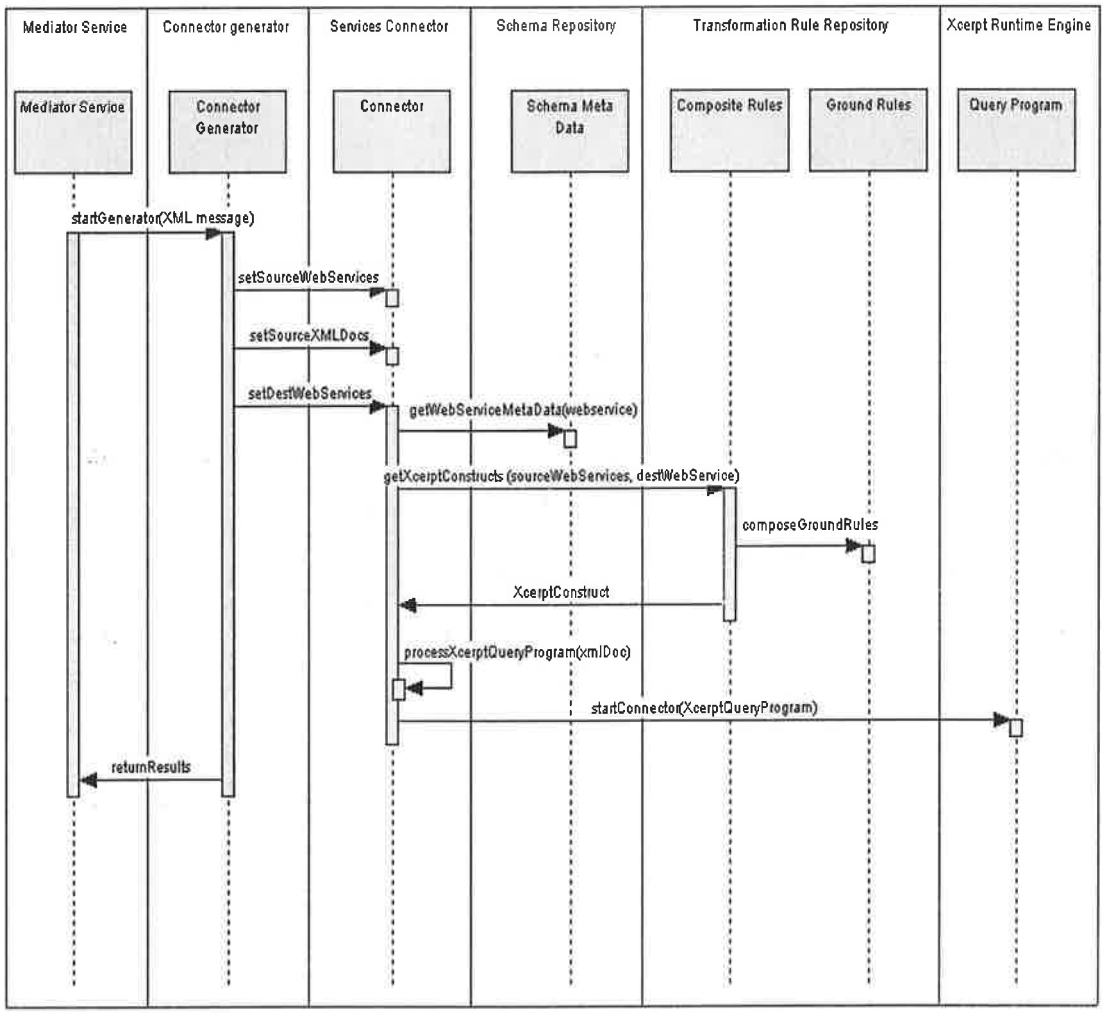


Fig. 5-3 The runtime interaction diagram of the connector generator.

The pseudo code in Fig. 5-4 illustrates the data flow among the components.

```
l_source_identifier1 :=
SetSourceWebService(webservice1);
l_source_identifier_array.add(l_source_identifier1)
;
l_source_xmldoc_array.add( setSourceXMLDocument
(l_source_identifier1, l_xml_doc1);
l_source_identifier2 :=
SetSourceWebService(webservice2);
l_source_xmldoc_array.add( setSourceXMLDocument
(l_source_identifier2, l_xml_doc2);
```

```
l_source_identifier_array.add(l_source_identifier2)
;
l_dest_identifier :=
SetSourceWebService(mediator_webservice1)
l_xcerpt_construct :=
getXcerptConstructs(l_source_identifier_array,
l_dest_identifier);
l_xcerpt_queryprogram := DynamicConnector
(l_xcerpt_construct, l_source_xmldoc_array);
l_desc_xml_doc := getTransformedXMLDocument
l_dest_identifier, l_xcerpt_queryprogram);
```

Fig. 5-4. The connector based on Xcerpt.

## 5.2 Enhancement and Adaptation of Xcerpt

We proposed to generate dynamically Xcerpt query programs to transform and mediate input SOAP messages from multiple Web services providers. However, Xcerpt is a document-centric language that is designed to query and transform XML documents or Web sites (Bry and Schaffert 2002). In Section 5.2.1, we investigate how Xcerpt can be integrated into the proposed SOA architecture. In Section 5.2.2, we look at how Xcerpt query programs can be dynamically constructed as connectors.

### 5.2.1 Resource Identifier Enhancement

Resources to be processed can be in the format of XML, HTML, RDF and even Xcerpt data terms in Fig. 5-5. The Xcerpt runtime engine reads resources and populates them into data terms before the query terms can start to evaluate them. The drawback is that all resources identifiers have to be specified inside a query program rather than be passed into a query program as parameters like ones in the conventional programming languages. As a result, the ground Xcerpt rules that read data from document resources are tied with file systems or a particular web site or location. The output of composite rules cannot be materialized until the execution of ground rules that a composite rule depends on populates the data terms. This is a bottom up approach in terms of data population because the data are assigned from the bottom level of the rules upward until it reaches a goal.

```
data Resource = XML URI
              | Xcerpt URI
              | HTML URI
              | Parsed Term
```

Fig. 5-5. Xcerpt resource specification.

If an external resource is omitted from a query term, Xcerpt by default consider that the resource specification is implicitly specified and therefore inherited from query terms at the lower level within a query program. If none of query terms has a resource specification in a query program, the resource for a query specification is the program itself. The example in Fig. 5-6 illustrates the Xcerpt syntax to specify output resources and input resources:

```
GOAL
      Out {
      Resource {"file:
      SupportIdentifier_Customer.xml"},
...
END
CONSTRUCT
      ...
FROM
      IN {
      Resource {
http://www.buyonline.com/getCustomer/customer.xml},
            ...
      }
END
```

Fig. 5-6 Resource identifiers in Xcerpt.

Xcerpt does not support automatic query program construction although it uses backward rule chaining technique to evaluate a chaining of queries. For example, Xcerpt does not allow properly distinguishing between Xcerpt constructs (e.g. variables) of the connector and Xcerpt constructs of stored descriptions. The query term in Fig. 5-7 expressed in XML format illustrates the problem. Xcerpt provides the functionality to convert its syntax into the XML representation.

```
<xcerpt:program xmlns:xcerpt="http://xcerpt.org">
    <xcerpt:construct>
      <nameAsContracted>
            <xcerpt:variable xcerpt:name="Name"/>
      </nameAsContracted>
</xcerpt:construct>
```

Fig. 5-7 The XML representation of Xcerpt construct term.

In this query term, it is unclear whether the variable "name" belongs to the generated connector or it is part of the stored queries. If it is a part of a connector, it needs to be bound to the element "`nameAsContracted`". Otherwise, the query should only match with rules that contain a construct part with a "`nameAsContracted`" element containing a variable named "`name`". There are

two solutions to this problem described in (Schaffert 2004). One is the use of quoting data, the other is the use of namespaces with which one namespace is for evaluated programs and use the resource URI of the queried program as the namespace for the queried program. The latter is a more elegant approach compared to the first one as namespace can clearly separate the variables from the evaluated programs to with the ones from the query programs.

### 5.2.2 The Xcerpt runtime environment

The Xcerpt runtime environment is implemented in Haskell (Wallace and Runciman 1999). The Xcerpt provides a command line interface (Schaffert 2004). In this section, we describe the runtime environment where the components collaborate to implement Xcerpt queries. The core of the Xcerpt runtime environment consists of the following five modules:

- **Data structure definer module**. This module defines data structures for data terms, query terms, goals, query programs, etc. It
- **IO module**. This module is responsible for retrieving data streams from local files or over the network and print out the output to the local files or onto the console.
- **Grammar parser module**. This module is responsible for parsing terms, programs, HTML and XML resources.
- **EngineNG module.** This module is at the core of the entire runtime environment. It is responsible for evaluating query programs by implementing various evaluation algorithms such as backward rule chaining, simulate unifications, substitution, and the constraint solver etc.
- **Methods module.** This module contains built-in functions such as comparisons, aggregations, and user-defined functions in the future.

We cannot simplify Web services as another form of XML databases or Websites in term of data access mechanism because no query language is currently available to transfer Xcerpt queries into SOAP messages. In the context of the ASP setting, Web service components responsible for the integration and mediation return the transformed XML document back to the mediated software architecture. The XML data flowing in and out of the Web service interfaces to the connector construction

component needs passing back into the proposed software architecture. However, the external resources passed into an Xcerpt query program cannot be identified with URIs. They are rather instances of XML documents generated at runtime. Furthermore, it is infeasible to save XML documents onto file systems where URIs can be assigned to these XML document for easy access because it requires the correlation of the input XML documents with the output XML document. It makes the XML documents very difficult to maintain and manage. Therefore, the current Xcerpt implementation is infeasible in the data integration solution because resource identifiers specified in the ground query terms are encapsulated within query programs. As a result, these resource identifiers are invisible to the connector construction component in our problem context.

In a summary, there are two problems in order to integrate Xcerpt into a Web service-based SOA architecture. One is to build up Xcerpt runtime to be a separate Web service within our SOA architecture, the other is to construct Xcerpt query programs dynamically to mediate and transform all the SOAP messages according to a shared mediated schema. The former problem requires building a Web service layer on top of Xcerpt runtime engine; the latter problem requires Xcerpt to take resources at the query program level rather than at the individual query level.

### 5.3 The proposed mediated software architecture

In this thesis, we investigate the heterogeneity of both data models and data formats in the mediated software architecture as described in Chapter 1. We propose, inspired by Haller (Haller et al. 2005 and Rosenberg (Rosenberg and Dustdar 2005), a service-oriented architecture for the data integration to provide a global view of data on demand from various data sources. The service-oriented data integration architecture is fundamentally different from business process integration as the latter is concerned with integrating business processes than data. The proposed integration architecture uses Web services to enable the provision of data on demand whilst keeping the underlying data sources autonomous. In this thesis, we concentrate on the connector-based data integration aspect. We propose that the generation of Xcerpt query program acting as connectors can be constructed dynamically at runtime. Other aspects of SOA such as security and access control are outside the scope of the thesis.

The design is to pass resource identifiers from the query program all the way down to the ground rules. By taking advantages of the evaluation process in Xcerpt, external resources can be loaded into a set of data terms by ground Xcerpt rules that subsequently can be queried by other rules that transform data according to the global structure. In the connector construction component, an Xcerpt query program is responsible for reading all the ground rules from the integration rules repository for the entire source Web service providers at runtime. Ground rules consumed by this query program are responsible for populating the XML data into the corresponding data terms. Another set of ground rules is the Xcerpt representation of the correspondences between the global XML schema and Web service providers. We identify these ground rules in advance for each of the Web service providers. In the Xcerpt representation of these ground rules, we use placeholders to replace the hard-coded resources identifiers. At runtime, firstly the connector construction component passes the XML documents from the mediators into the query program. Secondly, the designated query program reads all the ground rules for the involved source Web service providers. Thirdly, the placeholders in these ground rules are replaced with these XML documents held in the memory as illustrated in Fig. 5-8.

```
System.xcerpt: CONSTRUCT Var System
                 …
                FROM
                      IN {
                Resource {"%InputXmlDocument_for_System%"},
                           …
                      }
                END
                <-- After substitution, it becomes -->
                CONSTRUCT Var System
                      …
                FROM
                      IN {
          Resource {"<system>…</system>"},
                      …
                }
                END
```

Fig. 5-8 The resource substitution in Xcerpt.

At runtime, the generation of the query programs that transform the XML data and then the process of substitution will take place in the memory rather than loading the query program from the file system. The output XML document after data transformation is held in the memory rather than be saved on the file system.

We implement a java class that interacts with the Xcerpt runtime engine because the Xcerpt implementation has provided a Java API for the integration (Schaffert 2004). Then we adopt the Apache Web services Invocation framework (WSIF) (eLib: Apache Web Services Invocation Framework) to wrap the java class as a Web services rather than go through the soap stack. WSIF allows the java class and the Xcerpt runtime environment to remain independent of the details of the implementation of the Web services. What we need to do for the WSIF framework to know how to map from XML to Java is to define the WSDL file and the WSIF Binding for the java class. Finally, we need to deploy the Web service on a J2EE Application Server such as Oracle iAS (elib: Oracle Application Server).

The system architecture in Fig. 5-9 transforms the payload of SOAP messages into the format that conforms to a global schema. The data integration engine is built based on WS-BPEL, where invocation orders of data Web service providers are predefined in the integration schemas. Service orchestrations are defined by specifying the order in which operations should be invoked.

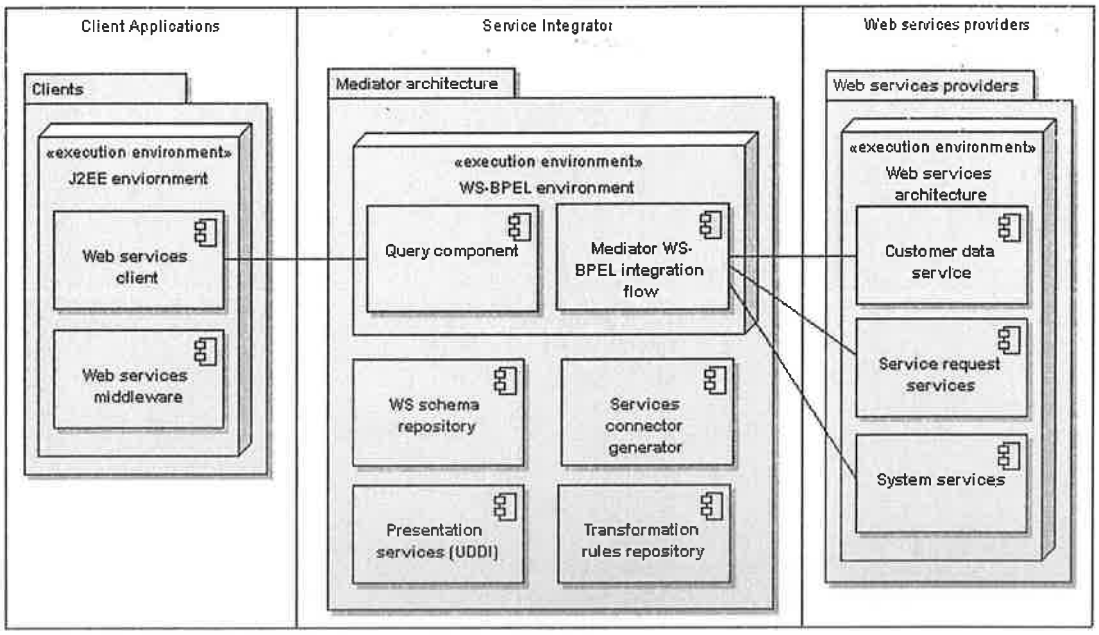


Fig. 5-9 The deployment view of a mediator Web services.

Each of these components in the architecture will be discussed from the point of view of data flow. To begin with, an XML message flowing from client applications through the architecture into data Web service providers, after the execution, the final output messages will flow back out to client applications. In the following

subsections, we introduce the components of our Web service-based mediator architecture in Fig. 5-9.

### 5.3.1 Client applications

Client Applications serves as a service requestor in the SOA. It triggers the request and normally asks for a response back. The Web service internal middleware that is normally generated by an IDE tooling such as Apache-AXIS is responsible for the communication with the Web service. In this case, it is the service integrator. The web service client is the component that makes the remote procedure calls published by the service integrator. Portal applications are a typical form of Web service clients. Wege (2002) defines that "a portal is a single, integrated point of access to information, applications and people. Portals integrate diverse interaction channels at a central point, providing a comprehensive context and an aggregated view across all information."

**Web services middleware** – Interactions are based on SOAP in Web services. Using SOAP, services can exchange messages by means of standardized conventions to turn a service invocation into an XML message, to exchange the message, and to turn the XML message back into an actual service invocation. Therefore, a Web service middleware is a SOAP-based middleware. The Web services middleware consists of a HTTP engine, a SOAP engine. The HTTP engine is responsible for sending XML messages over to the service integrator. The SOAP engine is responsible for packaging SOAP into HTTP and passes it to the HTTP engine.

**Web services client** – The Web services client are client stubs generated by the WSDL interfaces. The Web services client is responsible for providing the client applications to access XML messages as java class libraries so that client applications are able to invoke the services as local calls. Web Services Interoperability Organization (WS-I) has been established by industry leaders with the aim to promote interoperability of Web services implementations by publishing profiles. The WS-I profile consists of descriptions of conventions and practices for the use of Web services standards through which systems can interact such as versions and XML message encoding format. All Web service components in the architecture follow WS-I profile version 1.0 to limit interoperability related problems.

The component interactions described in this thesis are in synchronous and document-based fashion.

### 5.3.2 Service integrator

Exposing data sources as services is only the first step toward building a SOA solution. Other problems need to be addressed in order to benefit business such as global schemas and data transformations. Without the role of a service integrator, the service client needs to understand each of the data models and relationships of service providers.

- **Presentation service component**. The component is responsible for providing consumers an intuitive overview of what the information they are going to get from the mediator services by providing a Web interface. The component invokes the operations of each of the Web services from the UDDI directory directly via the UDDI APIs. The presentation service component enables potential consumers to invoke Web services operations from a browser in the hope that it might help them have a better understanding on the data exposed by the Mediator services.
- **Query component.** The query service is responsible for handling inbound requests from the application consumer side and outbound result back to client applications. WS-BPEL process flow container handles the internal messaging of the architecture. The application developers build their applications and processes around common objects and make successive calls to mediated Web services. The query component is also a Web service component providing a WSDL interface exposing all supported operations and services offered by mediator services. Therefore, interfaces of individual Web service providers are transparent to application customers; they may send any combinations of the input parameters to the query service. In order to facilitate these unpredicted needs, the query service has to decompose the input SOAP messages into a set of pre-defined WS-BPEL flows. Normally a WS-BPEL flow belongs to a mediator that delivers a single common object. Occasionally, two or more mediators need to be bundled together to deliver a single object.

- **Mediator services.** A mediator service is itself a WS-BPEL process flow. Mediators in our solution are used to deliver these data aspects according to the global schema. This schema is available to the customers also so that customers can decide which mediator to call based on the definition of the global schema. A mediator illustrated in Fig. 5-15 consists of the following components: the individual provided Web services, a WS-BPEL workflow, and one or more connectors. The focus of our study is not on the automatic composition of Web services, rather than on how the data output from multiple Web services can be integrated according to a global data model. Therefore, in terms of our WS-BPEL process flow, we can take a static approach with respect to the orchestration of the involved Web services. In our proposed architecture, the global data-overarching schema is maintained manually. The schema for large enterprise integration solutions may consist of various data entities. From the development point of view, it is only reasonable to deliver the mediator Web services on a phased basis such as one data aspect one release cycle.
- **Connector construction component.** This component is solely responsible for generating connectors for transforming messages both entering the WS-BPEL container from Web service providers and leaving the WS-BPEL container to Web service providers. It also transforms and merges messages from the WS-BPEL container back the Query component. This component is also the sole interface accessing the schema repository and transformation rule repository. The functionality of this component is two-fold: In terms of interactions between WS-BPEL container and Web service providers, it generates an adapter acting as an intermediary role to iron out the mismatch between messages. Most of WS-BPEL development IDE tools provide the similar functionality by providing wizards to guide the data mapping and generating XSLT underneath for the runtime execution. In terms of the data integration, it merges and transforms the temporary data from Web service providers according to a global data model.
- **Integration rule repository.** The core objective of repositories is to promote rule reuse to improve the maintenance of Xcerpt rules, and supporting multiple versions of Web service providers and mediator services.

- **WS Schema repository.** The repository stores the WSDL metadata and the XML schema information for both the Web service providers and the mediator Web services. The schema information will be used to validate the XML documents at runtime before they are integrated and returned to the client applications.

### 5.3.3 Web service providers

These providers provide source data retrieved from the underling data repositories to client and other services. Business domain analysts from both provider side and the data integration side agree to the signature of the Web service interfaces such as input parameter and data output in advance. The benefit of asking data sources to provide a Web service interface is to delegate the responsibility and cut down the effort spent on developing data access code and understanding the business logic.

### 5.3.4 Web services and WS-BPEL process container

We make use of Oracle Application Server with the integration of Oracle WS-BPEL Process Manager as our integration platform. It provides core support for Web services Discovery, Deployment and Service Invocation. A WS-BPEL process flow can be deployed into Process Manager as an independent Web Services component for service invocations.

## 5.4 The construction of mediator services

To illustrate how the construction of a mediator services works, we still use the use case given in Section 4.4.1. A global data model given in Fig. 5-10 has demonstrated the data relationship of the major business entities to define services delivered to customers. Each entity within the data model is stored in individual data repository at various locations within an organization. The full XML schema definition is given in Appendix A. Each data object is a logical representation of the entity and will often be populated with data sourced from more than one repository. For example, the customer object in the global data model needs pertinent data from both customer service request Web service provider and customer system Web service provider.

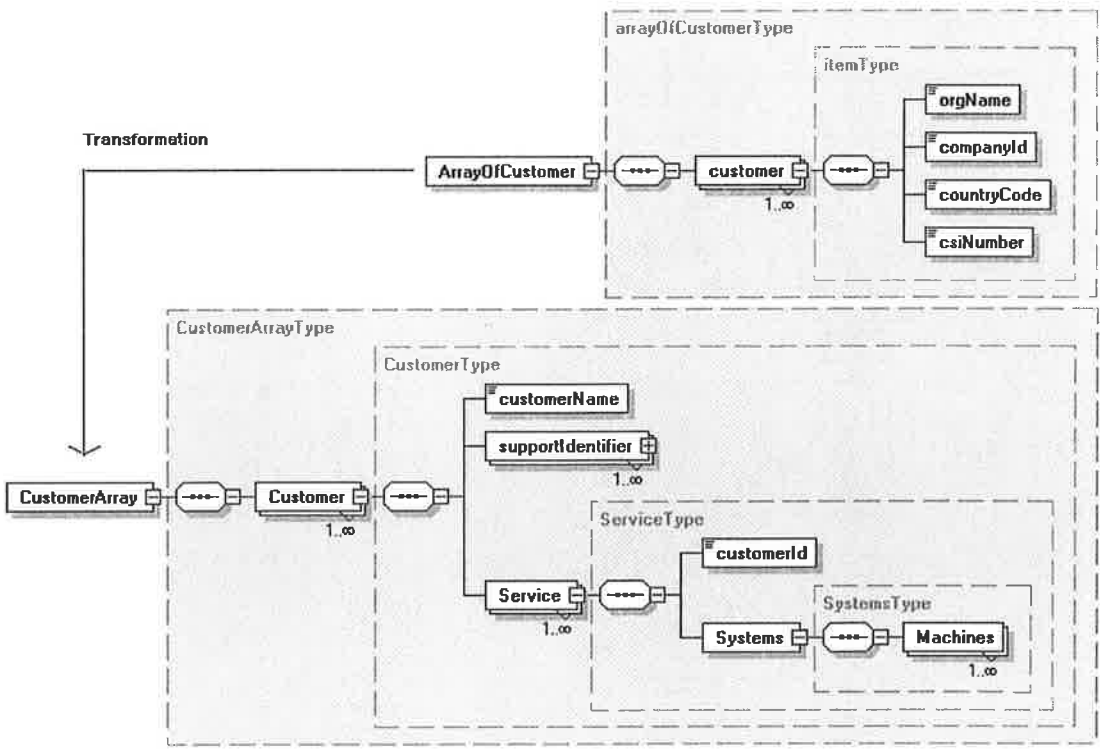


Fig. 5-10 The schema diagram of the global data model.

### 5.4.1 The WS-BPEL process flow in mediator Web services

In the proposed architecture, the global data model and the creation of rules are the responsibility of the business solution analysts, not necessarily the software architect. The rules are merely mappings from the elements exposed by Web service providers to the elements in the global data model. We have taken the approach in that business analysts determine the semantic similarity manually.

In the literature on data model transformation, the automation of the mapping is often limited to transforming the source model and the destination model rather than integrating more than more data models into a global data model. Even in the case of source to destination model mapping, the users attention is needed to select one from more than one sets of mapping that are generated (Rahm and Bernstein 2001). The quantitative work on accuracy of these schema mapping approaches are not addressed either. In our proposed architecture, the service connectors can be constructed dynamically by rule composition. The sacrifice is that semantic similarity is not taken into consideration.

The integration rules are created at the higher level than at the XML instance level as the schematic example in Fig. 5-11 demonstrates the integration rules governing

the integration process that produces the resulting XML data for the "Customer" mediator Web services. The output from the Customer mediator Web services represents a customer as identified in a servicing system. We list the Xcerpt query program in full in **Appendix B**.

```
Rules 1: This rule produces the "CustomerArray" by grouping
and reconstructing.
CustomerArray [all var customer,
                        all var supportidentifier,
                              all var services [
                                    var customerName,
                        all var system [[
                        var systemId,
                          all var machine
                                    ]]
                              ]]
                        ]]
                        ←
                        Customer [[
                              var customer,
                              var supportidentifier
                        ]]
                        ^
                        Service [[
                              var services [[
                                    var system [[ var machine]]
                              ]]
                        ]];

Rule 2: This construct rules get Customer data terms according
to the global data model.
Customer[[    var customer,
                              all var supportidentifier
                        ]]
                        ←
                        arrayOfCustomer[[
                              var customer,
                              var supportidentifier
                        ]];

Rule 2: This construct rules to get Service data terms
according to the global data model.
                        Service[[
var service [[
                              var system [[ var machine]]
                              ]]
                        ]]
                        ←
                        arrayOfService [[
                              var service [[
                                    var system[[ var systemId ]]
                              ]]
                        ]]
```

```
                    ^
                    Machine [[ var machine, var systemId]];

Rules 3: This construct rules to get Machine data terms
                Machines [[
                          all machine-of-system [[
                           var machine]],
                          var systemId
                          ]]
                ←
                machineItem [[
                      var machine,
                      var systemId
                ]];
```

Fig. 5-11 The composite rules in Xcerpt.

Each of the abovementioned rules is implemented in the Xcerpt language. In the above example, rule "`CustomerArray`" is a composite rule, based on "`Customer`" rule and "`Service`" rule that could be used to answer a users query directly. The resource identifiers in form of variables and the interfaces for the data representation will be supplied to the service connector generator. Rule mappings in the service connector generator determine which queries to be retrieved from the repository for execution. As a result, a query program including both query part and construction part is being executed to generate the XML output back to the service connector generator. We illustrate the corresponding activity diagram of one of WS-BPEL process flows in the "Customer" mediator Web services in Fig. 5-12.

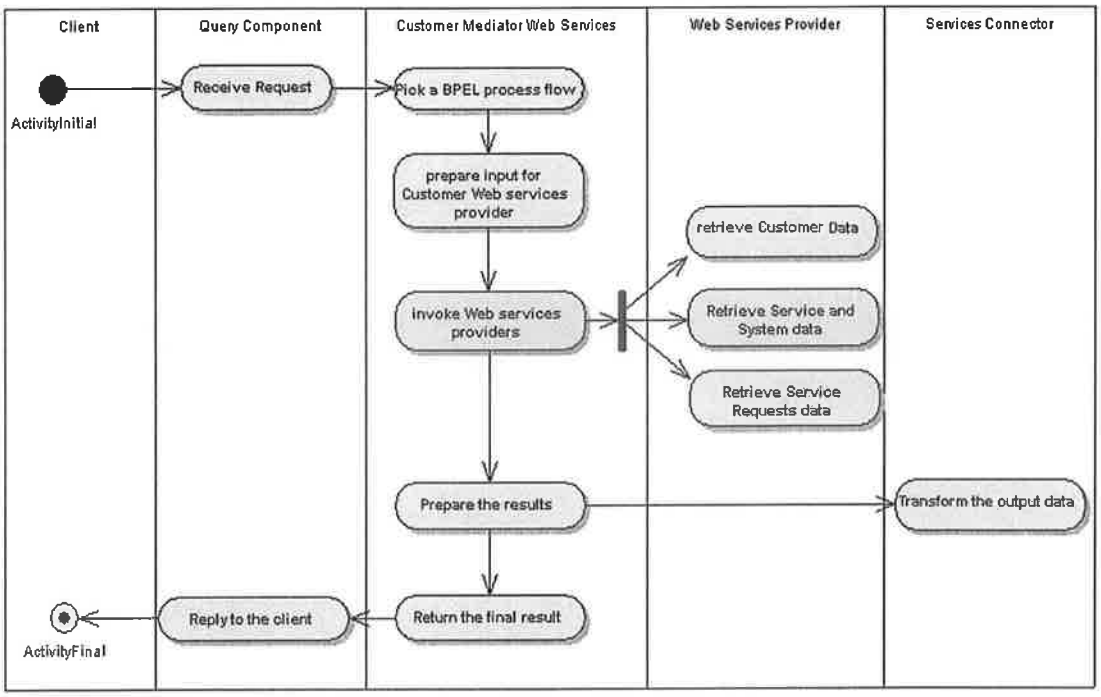

Fig. 5-12 The activity diagram of the WS-BPEL process flow.

We list WS-BPEL process flow in the XML representation in Fig. 5-13. All Web services that take part in the integration are registered in the WS-BPEL designer at design time as partner links. Each of Web service providers has a corresponding partner definition file where the XML schema of it is imported. The name and the role of "<partnerLink>" are defined in the locally managed partner definition file as described in **Appendix C**. The "<partnerLinks>" section in Fig. 5-13 defines parties that interact with the integration process in the course of processing the request of getting a customer's data. In this integration process there are six interacting roles: the client application, the mediator Web service, the service connector generator, the "Customers" Web service provider, the "Services and Systems" Web service provider and the "The Service requests" Web service provider. All the definitions for message types, ports type, operations are defined in the WSDL definition file in **Appendix C**.

```
<process name="CustomerMediator">
      <partnerLinks>
            <partnerLink name="CustomerMediatorWS_PL"
            myRole="CustomerMediator"/>
            <partnerLink name="ServicesConnectorGenerator_PL"
            partnerRole="ServicesConnectorGenerator"/>
            <partnerLink name="CustomerWSProvider_PL"
            partnerRole="CustomerWSProvider"/>
            <partnerLink name="ServicesAndSystemsWSProvider_PL"
            partnerRole="ServicesAndSystemsWSProvider"/>
            <partnerLink name="SRWSProvider_PL"
            partnerRole="SRWSProvider"/>
            <partnerLink name="QueryComponentWS_PL"
            partnerRole="QueryComponentWS"/>
      </partnerLinks>
      <variables>
            <variable name="inputGetCustomer"
            messageType="tns:GetCustomerRequestMessage"/>
            <variable name="outputGetCustomer"
            messageType="tns:CustomerResponseMessage"/>
      </variables>
      <!-- ORCHESTRATION LOGIC-->
      <sequence name="main">
      <pick name="pick-1">
      <onMessage partnerLink="client" operation="getCustomer"
      variable="inputGetCustomer">
            <sequence>
            <flow name="getCustomerFlow">
            <variables>
                  <variable name="inVar1"
                  messageType="service:getServicesReq"/>
```

```
      <variable name="outVar1" messageType="
      service:getServicesResp"/>
      <variable name="inVar2"
      messageType="cust:getCustProviderReq"/>
      <variable name="outVar2" messageType="
      cust:getCustProviderResp"/>
      <variable name="inVar3"
messageType="sr:getSRReq"/>
      <variable name="outVar3" messageType="
sr:getSRResp"/>
      <variable name="inServicesConnectorGenerator
      "
      messageType="connector:getServicesConnectorRe
      q"/>
      <variable name="outServicesConnectorGenerator
      "
      messageType="connector:getServicesConnector"/
      >
</variables>
<sequence name="getCustomer" >
      <invoke name="invoke-1"
      partnerLink="Customer_PL"
      operation="getCustomerContact"
      inputVariable="inVar2"
      outputVariable="outVar2" />
</sequence>
<sequence name="getServiceData">
      <invoke name="invoke-2"
      partnerLink="Services_PL"
      operation="getServicesData"
      inputVariable="inVar1"
      outputVariable="outVar1" />
</sequence>
<sequence>
      <invoke name="invoke-3"
      partnerLink="ServiceRequests_PL"
      operation="getSRData" inputVariable="inVar3"
      outputVariable="outVar3" />
</sequence>
</flow>
</sequence>
<sequence>
<assign name="assign-1-4">
<copy>
      <from variable="outVar1" part="payload"
      query="/('outVar1','return')/>
      <to variable="inServicesConnectorGenerator"
      part="payload"
      query="/tns:SourceXMLDoc/Customer"/>
      <from variable="outVar2" part="payload"
      query="/('outVar2','return')/>
      <to variable="inServicesConnectorGenerator"
      part="payload"
      query="/tns:SourceXMLDoc/Services"/>
      <from variable="outVar3" part="payload"
      query="/('outVar3','return')/>
```

```
                <to variable="inServicesConnectorGenerator"
                part="payload"
                query="/tns:SourceXMLDoc/ServicesRequest"/>
            </copy>
            <invoke name="invoke-4"
            partnerLink="ServicesConnectorGenerator_PL"
            inputVariable=" inServicesConnectorGenerator"
            otputputVariable="outputGetCustomer" />
            </sequence>
            <reply name="reply-1"
            partnerLink="QueryComponentWS"
            operation="getCustomer"
            variable="outputGetCustomer"/>
        </onMessage>
        </pick>
        </sequence>
</process>
```

Fig. 5-13 The WS-BPEL process flow for the "Customer" data.

We use XPath queries to select data expressions because we focus on the interactions between the mediator Web services and the Web service interfaces of the service connector generator. All XML messages returned from three Web service providers are copied into the XML messages that will be passed into the service connector generator Web services in the "`<assign name="assign-1-4"> ... </assign>`" section. The returned XML message will be passed back to the query component. The process flow in Fig. 5-13 in shown to comprise of an initial request from the client application, followed by an invocation of "Customer" Web service provider, "Services and Systems" Web service provider and "Service Requests" Web service provider in parallel, and ultimately a response to the client from the data sources sending the customer data.

In most of existing WS-BPEL integration solutions, XSLT transformation files are deployed together with the WS-BPEL Web services so that the WS-BPEL flow can refer to them at runtime in Fig. 5-14. Therefore, one XSLT transformation file can only be invoked by one WS-BPEL flow.

```
            <assign name="assign-1-5">
            <copy>
                <from
                expression="ora:processXSLT('xslt/getCustomer
                .xslt',bpws:getVariableData('outVar1','return
                ') )" />
                <to variable="outputGetCustomer"
                part="payload" query="/tns:CustomerObject"/>
            </copy>
```

```
</assign>
```

Fig. 5-14 The XSLT injection into WS-BPEL flow.

### 5.4.2 The interaction model of the architecture

In this section, we illustrate the interaction model in Fig. 5-15 with control flows among the components within a mediator Web service. Mediator services can also be integrated into a WS-BPEL process flow as data Web service providers.

The communications among components within a mediator Web services given in Fig. 5-15 works at runtime as follows:

1. Developers working for client applications generate the Web services client code based on the WSDL description published by the Query Component with any IDE tooling.
2. The Client Applications sent a remote method call to the Query Component.
3. The Query Component picks up an appropriate WS-BPEL process flow from the WS-BPEL Flow Repository based on the message matching at the interface level.
4. The chosen WS-BPEL process flow is executed in the WS-BPEL Flow container; The WS-BPEL Flow container is the runtime engine.
5. The WS-BPEL Flow Container calls the service connector generator for the message conversion if necessary at the beginning or in the middle of the execution in order to send the correct messages into a Web service next in the order.
6. The interim result returned from the complete execution of the Flow will be saved in the shared memory temporarily.
7. Once the execution of the entire flow is complete, all the results from a variety of Web services will be passed to the service connector generator for data transformation.
8. The merged data set will be returned to the WS-BPEL Flow to generate the output message.

9. The WS-BPEL Process Flow Web Service returns the desired result back to the Query Component.

10. The Query Component returns the desired result back to the client application.

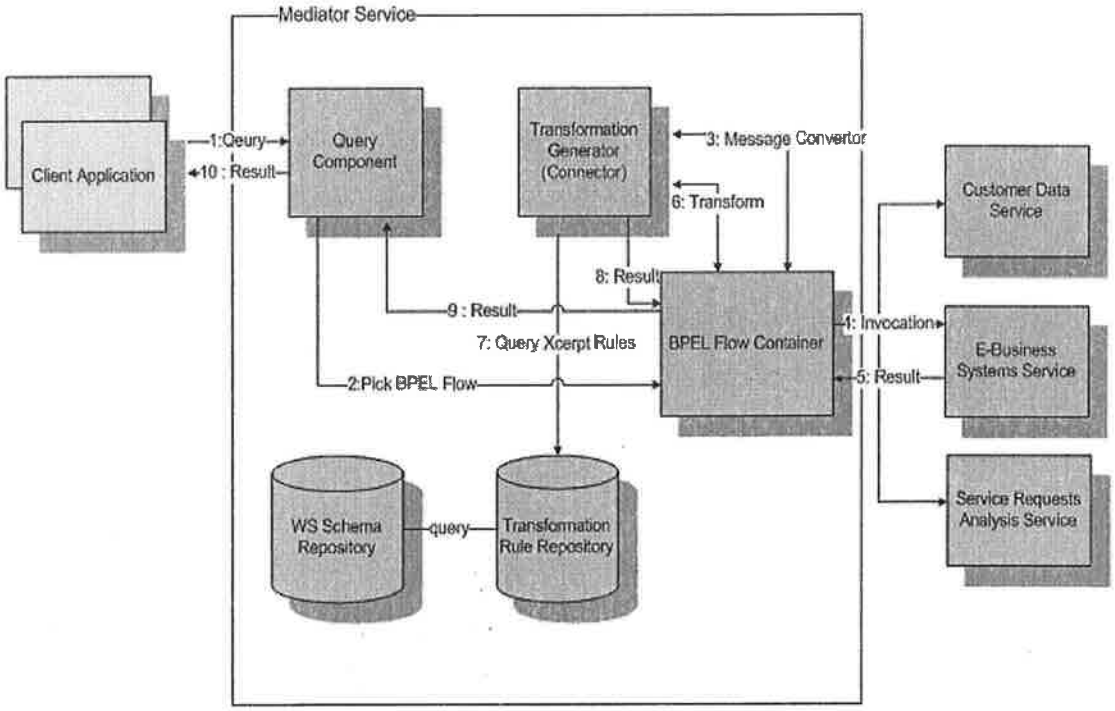


Fig. 5-15 The component model for the adaptive mediator Web service.

## 5.5 Summary

In this chapter, we have demonstrated how the connector construction component interacts with WS-BPEL flows in mediator Web services. In particular, we have described how the Xcerpt runtime environment is wrapped as Web services to be an integral part of the software architecture. Finally, we described the specification of the automatic construction of the Xcerpt query programs in the connector construction component that reads data integration rules from a repository.

# Chapter 6
# The evaluation of the proposed software architecture

In previous chapters, we have proposed a Web service-based SOA architecture in which a declarative rule-based data transformation technique have been integrated. The proposed architecture is tailored to address the problems pertaining to modifiability aspects of integration rules. In this chapter, we aim to evaluate the proposed software architecture with the scenario-based method with the goal of modifiability. In Section 6.1, we outline the concept of modifiability and approaches of evaluate modifiability in software architectures. In Section 6.2, we compare the proposed architecture with the XSLT based data transformation technique largely used in the commercial tools. In Section 6.3, we describe the results collected from the evaluation process.

## 6.1 Modifiability in software architecture

There are many approaches to evaluate modifiability in software architecture in the literature such as Architecture Tradeoff Analysis Method (ATAM) (Bass, Clements, and Kazman 2003), Architecture-level modifiability analysis (ALMA). These methods use scenarios to elicit change scenarios to evaluate the quality goal of a system. Software architecture evaluation can be conducted at various points in time during the software life cycle and with different goals in mind. In our case, we evaluate the architecture after the first release of the architecture has been implemented. We focus on how easy it is to modify the software systems to adapt changes in data integration rules. Authors in (Bengtsson et al. 2004) proposed the concept of modifiability, which focuses on the aspects of external changes rather than internal changes to the software systems such as bug fixing and corrections.

We have chosen the ALMA approach to evaluate the software architecture described in this thesis because of the following two reasons. Firstly, the modifiability aspect of the architecture is our essential objective. The ALMA approach specializes in evaluating the modifiability attribute of software architecture. Secondly, ALMA focuses on the modifiability aspect of the maintenance and distinguishes the analysis goals.

## 6.2 Evaluating modifiability in the proposed software architecture

In this section, we evaluate the proposed software architecture by following the five steps outlined in ALMA. In Section 6.2.1, we set up the goal for the evaluation process. In Section 6.2.2, we describe the architectural differences between the architectural candidates in view of component relationships. In Section 6.2.3, we elicit three change scenarios for evaluation.

### 6.2.1 Goal setting

An empirical approach is advisable to evaluate the architecture thoroughly. We are going to elicit scenarios that compare differences between the declarative rule-based approach and the existing software architectures using XSLT. We are also interested in predicting the development effect of building data integration and mediation component.

In previous chapters, we have demonstrated the modifiability of the proposed architecture by reengineering legacy systems into Web service-based components. Web services technologies help eliminate the mismatches between the interfaces exposed by the software components and the interfaces that are required in the new integration practice. We identify three change scenarios in terms of requirement changes. Firstly, changes to Xcerpt integration rules led by changes of business rules. Secondly, changes to ground rules led by cosmetic changes of the data model of Web service providers. Thirdly, introducing new Xcerpt integration rules led by new entities added into the global data model. In Section 6.2.3, we evaluate each of the change scenarios in detail.

### 6.2.2 Architecture description

This thesis is motivated in the context of application service providing with service-oriented architectures. In this context, appropriate data transformation is required. We have proposed a new data integration technique to solve the problems related to modifiability of rules that governing data integration rules into Web service-based SOA. The contribution of this thesis is in the area of XML transformation, we therefore compare the proposed architecture with traditional XSLT architectures in this section. Currently we can only evaluate the architecture by predication from our contributions. We need descriptions of both to compare two software architectures. Architecture-level impact analysis is to identify either architectural element affected

by a change scenario directly or indirectly (Bengtsson et al. 2004). The component model of mediator Web service in the proposed software architecture is given in Fig. 5-15. In this section, we focus on decomposing the proposed software architecture into components, connectors, and their relationships. Properties of architectural components can be represented as class attributes or with associations using UML models (Bass, Clements, and Kazman 2003). In Fig. 6-2, we use no explicit representations to model interfaces and use UML classes for connectors and components to model the component-and-connector view of the proposed software architecture. In Fig. 6-1, we describe the decomposition and the relations of the components using the same UML notations as in Fig. 6-2.

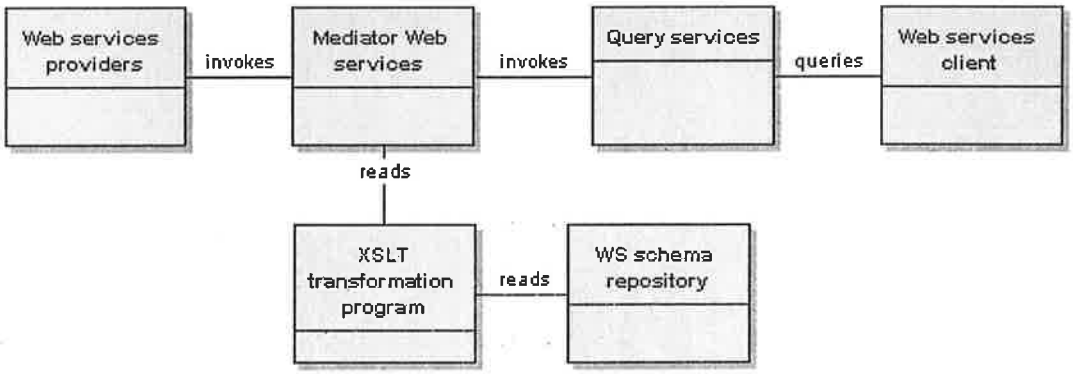


Fig. 6-1 The current software architecture injected with XSLT transformation files.

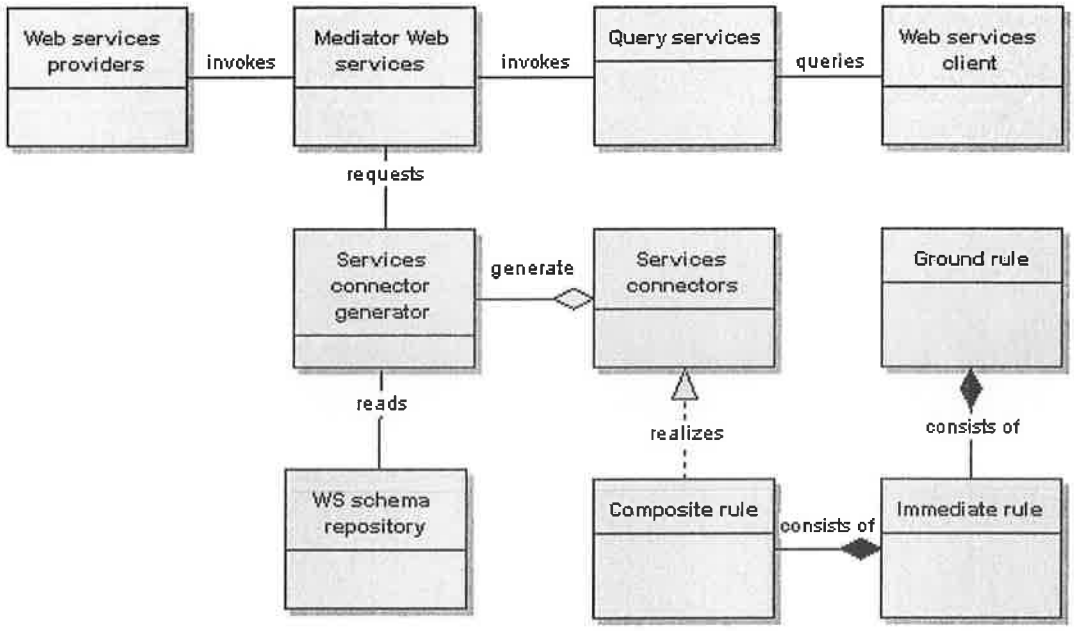


Fig. 6-2 The component-and-connector view of our software architecture.

### 6.2.3 Change scenario elicitation

Within the development process, in-house software architects perform internal assessments while external experts perform inspections on the software architecture. It would be ideal if we could conduct a long-term empirical study of modifiability of the proposed software architecture. The approach that we take is to elicit scenarios and evaluate these scenarios. In this section, we identify the following three scenarios for evaluations.

**Changes to integration rules at the mediator level** – Business rules change more often than the data model (Rouvellou et al. 2000). Our approach is to have various mediator Web services rather than only one to support the fully specified unified virtual view. The unified virtual view supports an incremental definition as more mediator Web services are built to answer users' queries. Indeed, the unified virtual view is subject to changes and additions as the analysis of the information sources proceeds.

**Changes to initial correspondences at the element level –** Two scenarios may happen in terms of changes of ground rules. One is that source attributes of a data mapping from one Web service provider from one to another; the other is the name of attributes of a data object in the unified data model changes or the ones in the data model of a Web service provider change.

**Adds new integration rules –** This scenario might happen on two occasions: one is that customers request new mediator Web services, the other is that a new Web service provider comes on board as the service integrator has gained its influence and reputations. Nonetheless, it will introduce new integration rules at the top level of the global data model.

## 6.3 Change scenarios evaluation and interpretation

We evaluate the scenarios by the methodology that for each scenario, we determine the architectures for comparison that support better in terms of how many components are affected directly or indirectly. We express the results as a list of the scenarios with the better architecture for each scenario in Table 6-1.

In the first change scenario, as mentioned in Chapter 1, one of the problems that the traditional data transformation languages XSLT have is that a set of business rules to render unified entities in the global data model are intertwined among each

other although the business rules are separated from the application logic. In our software architecture, we have implemented the declarative rule-based approach in which the rules for integration are represented in Xcerpt and saved separately in the rule repository. The business rules are composed together at runtime during the construction of the service connectors by the service connector generator. Therefore, changes of business rules do not affect the rest of the business rules because they are rather independent. On the contrary, the query part and the construct part of the XSLT transformation queries are tightly coupled. Therefore, it is almost impossible to automate the construction process of a transformation file.

The second change scenario relates to the changes of a data term when one of source XML documents is replaced with a new source. It only affects the population of one data term. In the case of changes to the names of attributes on both sides, the data terms remain untouched, then the query terms do not need to change either. The only place to change is the construct term of a business rule. For example, one of the data terms at the lower level changes will only need to update the directly referenced query term, the construct term in an integration rule will not be affected at all. In order to handle the same scenario for XSLT, it also means to construct up another new version of entire XSLT transformation file.

In the third changing scenario, the immediate composite rules can be leveraged from the existing ones since possibilities are that the new Web service provider might share some common entities. This scenario also demonstrates that the modifiability of the Xcerpt integration rules can occur on the following two scenarios: one is to build the new mediator Web service, the other is to build a new version of the mediator Web service.

Table 6-1 The results of the evaluation.

| **Scenarios** | **Proposed software Architecture** | **Existing architectures and commercial tools** | **How achieved or tactics used in the proposed software architecture** |
|---|---|---|---|
| Change scenario 1 | Composite rules | Entire XSLT transformation file | Automatic program construction at runtime |
| Change scenarios 2 | Ground rules, maybe | Entire XSLT | The query part and the |

| | | | |
|---|---|---|---|
| | some immediate rules | transformation file | construct part of an integration rule are separately expressed |
| Change scenario 3 | New version of composite rules, or reuse or add ground rules and immediate rules. | Entire XSLT transformation file | The integration rule repository and the independent services The connector generator component inject no code into the integration flow. |

## 6.4 Summary

In this chapter, we have evaluated the proposed software architecture following the ALMA evaluation techniques. We have compared the proposed software architecture with the existing architectures using XSLT data transformation files that are injected into the WS-BPEL integration flows. We have elicited three typical changing scenarios to assess the modifiability in the software architectures. The results have suggested that the proposed software architecture does improve the modifiability using the declarative rule-based approach. However, in order to complement this evaluation, a long-term empirical study of modifiability of the proposed software architecture would be ideal, but it is not possible because the time constraints on this research.

# Chapter 7
# Conclusions

This thesis investigated the use of Web service-based SOA approach to reduce heterogeneity and interoperability of the traditional data integration problems. In order to solve challenges for data integration and mediation for XML data in Web service-based SOA, we had assessed query and transformation language Xcerpt and its data integration techniques such as backward rule chaining and the simulation unification. We have determined that Xcerpt can be adapted to maintain the loosely couple nature of the proposed software architecture and to improve the transformation code modifiability and reuse. We have proposed that the automation of the construction of data mediation as connectors can be used to connect Web service providers and integrate the XML output according to a global data model. We have also proposed a mediator-based data integration architecture where the Xcerpt-based connectors are integrated with WS-BPEL-based Web services. From the evaluation of the architecture, we have proved that the automation of the Xcerpt-based connectors can improve the modifiability of integration rules that govern the data integration flow in software architecture.

## 7.1 Summary of contribution

In this thesis, we have followed a systematic approach to select the most suitable language for the data integration and mediation of the XML data into our solutions. Firstly, we have defined a set of selection criteria based on our problems specific requirements and the existing ones in the literature. Secondly, we have evaluated a selection of language candidates following these criteria. Finally, we have adopted and adapted Xcerpt, a declarative rule-based XML query and transformation language to build service connectors for the mediator Web services.

We have determined that declarative rule-based approach can be adapted to automate a part of data integration process in the Web service-based software architecture. The major differences between the declarative rule-based approach using Xcerpt with others conventional data integration and transformation approaches using XSLT or XQuery are the language constructs and data integration technique that Xcerpt has brought to us. The support of the separate representation of

the query part and the construct part of an integration rule in Xcerpt makes the automation of a query program possible in a data integration architecture. The automated connector can be adopted to both connect the selected Web service provider within a process flow and integrate the XML output of each of the Web service providers afterwards in order to render the data objects according to a global data model.

## 7.2 Future research

The goal of SOA realized by Enterprise Services Bus (ESB) (Chappell 2004) is to discover and assemble Web services on demand. ESB architecture functions as both transportation and mediation facilitator to allow distribution of these Web services over disparate data sources. Therefore, a data integration system in SOA could provide more value if Web service providers are dynamically discoverable and able to access queries of an open standard query language. The proposed WS-BPEL-based mediator Web services can act as the mediation layer and information aggregation component in ESB to facilitate message routing and message mediation. However, Web services in ESB architectures can be numerous as opposed to ones internal to an enterprise. Therefore, the automation of identification of data integration and mediation rules and the automated Web service assembly in WS-BPEL might worth investigating. We outline a few areas that we have identified that have room for improvement in order to be adopted in broader application domains. In Section 7.2.1, we suggest that the Semantic Web technologies can be used to detect the semantic similarity for the automatic construction of the WS-BPEL integration flows. In Section 7.2.2, we discuss the possibility of writing data back to Web service providers. In Section 7.2.3, we discuss the possibility of adding security and access control into the proposed software architecture.

### 7.2.1 The semantic similarity

In our current approach, execution plans to answer user's requests are pre-defined in WS-BPEL and saved in the repository because the focus of this thesis is on the automation of the connectors related to data integration and mediation aspects of SOA. We believe that the execution plans can be automated by utilizing the Semantic Web technologies. Arguably, the current Web service standards do not have the reasoning power and semantics to be discovered and composed

automatically. Many researches have been proposed towards this direction such as WSMX (Haller et al. 2005). WSMX is a reference execution environment for Semantic Web services where similar data mediators are used to solve interoperability problems because Web service uses different ontology. Another approach to the semantic composition of Web services is the ontology-based composition. The ontological frameworks for service process composition have been proposed in (Pahl and Zhu 2005, King and Roantree 2005) has provided the service process ontology to construct WS-BPEL process flow from the automatic assembly of Web services. Therefore, it is worth investigating to automate the construction of the WS-BPEL integration flow with the support of the WS-BPEL service process ontology.

### 7.2.2 Writing data back to the mediator Web services

Schema integration in DBMS has addressed the problem of view updates. An adapter defines the view that can be manipulated so that changes are reported back to the source. This is not required for the document-centric integration in this research in the ASP domain. However, it may be required when the proposed software architecture is adopted to solve data integration problems in other problem domains. The main challenge in writing data aback to the mediator Web services is to translate the queries imposed on the mediator Web services into subsequent queries that can be understood by the Web service providers. The Service Data Objects (SDO) (Castro 2004) provides a unified data model for retrieving and updating data from data sources including Web service providers. However, SDO specification needs two-phased transaction support from the architecture to commit transactions across affected data sources. Therefore, the proposed software architecture needs the extension of supporting transactions in order to enable writing data back to the mediator Web services.

### 7.2.3 Security and access control

In the proposed software architecture, the security and the access control aspects are out of the scope of the work because our problem context is an ASP setting where data sources that need integration and mediation are internal to client applications. However, the Web services security and access control are worth investigating in order that the proposed software architecture has a wider adoption such as in ESB

architectures. An identification service can be plugged into the architecture if the Web service providers trust the proposed architecture to carry out security and identification functions for them. The identification service component provides protection to mediator web services by checking up the signature of calls and Web service providers by checking up the identity of the caller that is assigned by the identify services component.

# Bibliography


Abiteboul, S. Cluet, S. Milo, T. 1997. Correspondence and Translation for Heterogeneous Data. *IN: Conf. on Database Theory*, 1997. Essex: Elsevier Science Publishers. pp179-213.

Abiteboul, S. 1997. Querying semi-structured Data. *IN: Proceedings of 6th International Conference on Database Theory.* January 1997. Delphi: Springer. pp1–18.

Abiteboul, S, Buneman, P. Suciu, D. 2000. *Data on the Web. From Relations to Semistructured Data and XML.* Berlin : Morgan Kaufmann.

Abiteboul, S. Benjelloun, O. Milo, T. 2002. Web services and data integration. *IN: Proceedings of the Third International Conference on Web Information Systems Engineering,* December 2002. pp3-6.

Alonso, G. Casati, F. Kuno, H. Machiraju, V. 2004. *Web Services – Concepts, Architectures and Applications*. New York: Springer Verlag.

Andrews, T. Curbera, F. Dholakia, F. Goland, Y. Klein, J. Leymann, F. Liu, K. Roller, D. Smith, D. Thatte, S. Trickovic, I. Weerawarana, S. 2003. Business Process Execution Language for Web Services Version 1.1 [Online]. Available from : <http://www.ibm.com/developerworks/library/ws-bpel/> [Accessed 10 September, 2006].

Banerji, A. Bartolini. C. Beringer, D. Chopella, V. Govindarajan, K. Karp, A. Kuno, H. Lemon, M. Pogossiants, G. Sharma, S. Williams S. 2002. Web Services Conversation Language (WSCL) [Online]. Available from : <www.w3.org/TR/2002/NOTE-wscl10-20020314/> [Accessed 10 September 2006].

Bass, L. Clements, P. Kazman, R. 2003. *Software Architecture in Practice*. 2nd Edition. Boston: Addison-Wesley.

Bengtsson, P. Lassing, N. Bosch, J. Vliet, H. 2004. Architecture-Level Modifiability Analysis (ALMA). *Journal of Systems and Software*, 69 (1), pp129-147.

Benzaken, V. Castagna, G. Frisch, A. 2003. CDuce: An XML-Centric General-Purpose Language. *IN: Proceedings of the ACM International Conference on Functional Programming*, 2003. Uppsala: ACM Press. pp51-63.

Berlea, A. Seidl, H. 2001. fxt A Transformation Language for XML Documents. *IN: Proceedings of XML Conference and Exposition*, December 2001. Orlando.

Berger, S. Coquery, E. Drabent, W. Wilk, A. 2005. Descriptive Typing Rules for Xcerpt. *IN: Proceedings of International Workshop, PPSWR*, September 2005, Dagstuhl Castle Springer Verlag.

Boag, S. Chamberlin, D. Fernndez, M. Florescu, D. Robie, J. Simon, J. 2004. XQuery 1.0: An XML query language [Online], Available from : < http://www.w3.org/TR/xquery/ > [Accessed 04 March 2006].

Bolzer, M. 2005. *Towards Data-Integration on the Semantic Web: Querying RDF with Xcerpt*. Master Thesis. University of Munich.

Bonifati A. Ceri, S. 2000. Comparative analysis of five XML query languages. *SIGMOD Record*, 29 (1), pp68-79.

Booth, D. Haas, H. McCabe, F. Champion, M. Ferris, C. Orchard, D. 2002. Web Services Architecture [Online]. Available from : < http://www.w3.org/TR/ws-arch/> [Accessed April 19, 2006].

Boukottaya, A. Vanoirbeek, C. 2005. Schema Matching for Transforming Structured Documents. *IN: Proceedings of the 2005 ACM symposium on Document engineering*. New York: ACM Press. pp101-110.

Bry, F. Schaffert, S. 2002. A Gentle Introduction into Xcerpt, a Rule-based Query and Transformation Language for. *IN: RuleML Workshop at the International Semantic Web Conference*, June 2002. Sardinia.

Bry, F. Schaffert, S. 2002. Towards a Declarative Query and Transformation Language for XML and Semistructured Data: Simulation Unification. *IN: Proceedings of the 18th International Conference on Logic Programming (ICLP)*, LNCS 2401, July 2002.

Bry, F. Schaffert, S. 2002. The XML Query Language Xcerpt: Design Principles, Examples, and Semantics. *IN: Workshop on Web Databases at NETObjectDays'02* - LNCS 2593, October 2002. Erfurt.

Buneman, P. Fernandez, M. Suciu, D. 2000. UnQL: A Query Language and Algebra for semistructured Data Based on Structural Recursion. *VLDB Journal*, 9 (1), pp76-110.

Calvanese, D. Giacomo, G, Lenzerini, M. Nardi. D. 2001. Data Integration in Data Warehousing. *International Journal of Cooperative Information Systems*. 10 (3), pp237-271.

Carey, M. Haas, L. Schwarz, P. Arya, M. Cody, W. Fagin, R. Flickner, M. Luniewski, A. Niblack, W. Petkovic, D. Thomas, J. Williams, J. Wimmers, E. 1995. Towards heterogeneous multi-media information systems: The Garlic approach. *IN: Proceedings of the fifth Int. Workshop on Research Issues in Data Engineering – Distributed Object Management.* pp124–131.

Carey, M. 2006. Data delivery in a service-oriented world: the BEA AquaLogic data services platform. *IN: Proceedings of the 2006 ACM SIGMOD international conference on Management of data,* 2006. New York: ACM Press. pp695-705.

Castro, P. Giraud, F. Konuru, R. Purakayastha, A. Yeh, D. 2004. A programming framework for mobilizing enterprise applications. *Sixth IEEE Workshop on Mobile Computing Systems Applications*, 2004. pp196-205.

Chamberlin, D. Fankhauser, P. Marchiori, M. Robie, J. 2005. XML Query Use Cases [Online]. Available from : < http://www.w3.org/TR/xquery-use-cases/> [Accessed 12 September 2006].

Chappell, D. 2004. *Enterprise Service Bus*. Sebastopol : O'Reilly Media, Inc.

Clark, J. 1999. XSL Transformations (XSLT) Version 1.0 [Online]. Available from : < http://www.w3.org/TR/xslt >. [Accessed 01 September 2006].

Cluet, S, Delobel, C. Simeon, J. Smaga, K. 1998. Your mediators need data conversion! *IN: Proceedings ACM SIGMOD International Conference on Management of Data,* 1998. Washington : ACM Press. pp177-188.

Crnkovic, I, Larsson, M. 2000. A Case Study: Demands on Component-based Development. *IN: Proc. 2nd International Conference on Software Engineering*. ACM Press. pp23-31.

Deutsch, A. Fernandez, M. Florescu, D. Levy, A. Suciu, D. 1998. XML-QL: A Query Language for XML. *IN: Proc. W3C QL'98 – Query Languages 1998.*

Doan, A, Domingos, P. and Halevy, A. 2001.Reconciling schemas of disparate data sources: a machine-learning approach. *IN: Proceedings of ACM SIGMOD Conference*, 2001. pp509–520.

Doan, A, Halevy, A. 2005. Semantic Integration Research in the Database Community: A Brief Survey. *AI Magazine, Special Issue on Semantic Integration*, Spring 2005.

Eisenberg, E. Melton, J. 2004. An early look at XQuery API for Java™ (XQJ). *ACM SIGMOD Record*, 33 (2), pp105-111.

eLib: Apache Axis. (Homepage). [Online]. Available from : <http://ws.apache.org/axis/> [Accessed 24 August 2006].

eLib: Apache Web Services Invocation Framework [Homepage]. [Online]. Available from : <http://ws.apache.org/wsif/> [Accessed 21 September 2006].

eLib: Apache ServiceMix. (Homepage). [Online]. Available from : <http://servicemix.org> [Accessed 08 August 2006].

eLib: OMG's MetaObject Facility (MOF). (Homepage). [Online]. Available from : <http://www.omg.org/mof/> [Accessed 16 August 2006].

elib: Oracle Application Server. (Homepage). [Online]. Available from : <http://www.oracle.com/appserver/index.html > [Accessed 07 June 2006].

eLib: Oracle WS-BPEL Process Manager 2005. (Homepage). [Online]. Available from : <www.oracle.com/technology/products/ias/bpel/index.html> [Accessed 26 February 2006].

eLib: Oracle Jdevleoper (Homepage). [Online]. Available from : < http://www.oracle.com/technology/products/jdev/index.html > [Accessed 12 July 2006].

Frankel, D. 2003. *Model Driven Architecture: Applying MDA to Enterprise Computing*. Indianapolis : Wiley.

Garcia-Molina, H. Papakonstantinou, Y. Quass, D. Rajaraman, A. Sagiv, Y. Ullman, D. Vassalos, V. Widom, J. 1997. The TSIMMIS approach to mediation: Data models and languages. *Journal of Intelligent Information Systems*, 8 (2), pp117-132.

Groppe, S. Bottcher, S. 2003. XPath query transformation based on XSLT stylesheets. *IN: Proceedings of the 5th ACM international workshop on Web information and data management.* New York: ACM. pp106-110.

Gruninger, M. Lee J. 2002. Ontology applications and design. *Communications of the ACM*, 45(2), pp39-41.

Hay, D. Kealy, K. 2000. "Defining Business Rules - What are they really?" the final report of the "GUIDE Business Rules Project" [Online]. Available from : < http://www.businessrulesgroup.org/first_paper/BRG-whatisBR_3ed.pdf > [Accessed 29 August 2006].

Haller, A. Cimpian, E. Mocan, A. Oren, E. Bussler, C. 2005. WSMX – a semantic service oriented architecture. *IN: Proceedings of international conference on Web services.* 2005. Orlando.

Hasselbring, W. 2002. Web data integration for e-commerce applications. *IEEE Multimedia.* 9 (1). pp16-25.

Heimbigner, D, Mcleod, D. 1985. A federated architecture for information management. *ACM Transactions on Information Systems (TOIS).* 3 (3), 253-278.

Hosoya, H, Buneman, P. 2003. XDuce: A Typed XML Processing Language. *ACM Trans. Internet Techn.* 3 (2), pp117-148.

Katz, H. 2004. *XQuery from the Experts.* Boston : Addison-Wesley.

Kavantzas, N. Burdett, D. Ritzinger, G. 2004. Web Services Choreography Description Language Version 1.0 [Online]. Available from : <http://www.w3.org/TR/ws-cdl-10/> [Accessed 19 September 2006].

King, N. Roantree, M. 2005. Process Composition Using A Semantic Registry. *International Workshop Data Integration and the Semantic Web (DISWeb'05)* June 14, 2005, Porto.

Lehti, P. Fankhauser, P. 2004. XML data integration with OWL: experiences and challenges. *IN: Proceedings. 2004 International Symposium on Applications and the Internet.* pp160-167

Lenzerini, M. 2002. Data integration: A theoretical perspective. *IN: Proceedings of the twenty-first ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, 2002. Madison: *ACM.* pp233-246.

Levy, A. 1998. The information manifold approach to data integration. *IEEE Intelligent Systems*. pp1312-1316.

Leymann, F. 2001. Web Services Flow Language (WSFL 1.0) [Online]. Available from : <http://www-3.ibm.com/software/solutions/webservices/pdf/WSFL.pdf/> [Accessed 19 April 2006].

Ludoscher, B. Papakonstantinou, Y. Velikhov, P. 1999. A Brief Introduction to XMAS. Technical Report. Database Group at University of California, San Diego, 1999.

Madhavan, J. Bernstein, A. Rahm, E. 2001. Generic Schema Matching with Cupid. *VLDB*. pp44-58.

Maier, D. 1998. Database Desiderata for an XML Query Language. *IN: Proceeding of W3C Query Languages* 1998, December 1998.

Martin, D. Burstein, M. Hobbs, J. Lassila, O. McDermott, D. McIlraith, S. Narayanan, S. Paolucci, M. Parsia, B. Payne, T. Sirin, E. Srinivasan, N. Sycara, K. 2004. OWL-S: Semantic Markup for Web Services. [Online]. Available from : <http://www.daml.org/services/owl-s/1.0/owl-s.html> [Accessed 25 September 2006].

Milanovic, N. Malek, M. 2004. Current solutions for Web service composition. *IEEE Internet Computing,* 8 (6), pp51-59.

Milner, R. Tofte, M. Harper, R. MacQueen, D. 1997. *The Definition of Standard ML (Revised).* MIT Press.

Miller, R. Haas, M. Hernandez, M. 2000. Schema mapping as Query Discovery. *IN: Proceeding of the International Conference on VLDB*, Cairo: VLDB, pp77-88.

Milo, T, Zohar, S. 1998. Using Schema Matching to simplify heterogeneous Data Translation. *IN: Proceeding of the Int'l Conf.* New York: VLDB. pp122-133.

OMG. 2005. MOF QVT Final Adopted Specification [Online]. Available from : <http:// www.omg.org/docs/ptc/05-11-01.pdf> [Accessed 04 June 2006].

Orriens, B. Yang, J. Papazoglou, M. 2003. A Framework for Business Rule Driven Web Service Composition. *IN: ER 2003 Workshops*, 2003. Berlin: Springer-Verlag. pp52-64.

Pahl, C. 2002. A Formal Composition and Interaction Model for a Web Component Platform. *IN: Proc. ICALP Workshop on Formal Methods and Component Interaction.* A. Brogi and E. Pimentel, ed(s). Elsevier Electronic Notes in Theoretical Computer Science.

Pahl, C. Zhu, Y. 2005. A Semantical Framework for the Orchestration and Choreography of Web Services. *IN: International Workshop on Web Languages and Formal Methods WLFM'05.* Newcastle. Elsevier ENTCS Series. 2005.

Papakonstantinou, Y. Abiteboul, S. Garcia-Molina H. 1996. Object Fusion in Mediator Systems. *IN: Proceedings of the 22 International Conference on Very Large Database, 1996 [*Online*].* Available from : <http://www.db.ucsd.edu:8080/root/pubsFileFolder/147.pdf > [Accessed 22 August 2006].

Papakonstantinou, Y and Garcia-Molina, H and Ullman, J. 1996. MedMaker a mediation system based on declarative specifications. *IN: Proceedings of the 12th International Conference on Data Engineering,* February, *1996.* IEEE Computer Society. pp131-141.

Papakonstantinou, Y. Velikhov, P. 1999. Enhancing semistructured data mediators with document type definitions. Data Engineering. *IN: Proceedings 15th International Conference,* March 1999. pp136-145.

Peltier, M. B´ezivin, J. Guillaume, G. 2001. MTRANS: A general framework, based on XSLT, for model transformations. *IN: Proceedsings of the Workshop on Transformations in UML,* Apr. 2001.Genova.

Rahm, E. Bernstein, A. 2001. A Survey of Approaches to Automatic Schema Matching. *VLDB Journal,* 10 (4). pp334-350.

Reynaud, C. Sirot, P. Vodislav, D. 2001 Semantic Integration of XML Heterogeneous Data Sources. *IN: Proc. IDEAS,* 2001. pp199-208.

Rosenberg, F. Dustdar, S. 2005. Business Rules Integration in BPEL - A Service Oriented Approach. *IN: Proceedings of the 7th International IEEE Conference on E-Commerce Technology.* Munich.

Rouvellou, I. Degenaro, L. Rasmus, K. Ehnebuske, D. McKee, B. 2000. Extending business objects with business rules. *IN: Proceedings of the 33rd International Conference on Technology of Object-Oriented Languages*. pp238-249.

Schaffert, S. 2004. *Xcerpt: A Rule-Based Query and Transformation Language for the Web*. PhD Thesis, University of Munich, October 2004.

Sheth A. P. Larson A. 1990. Federated database systems for managing distributed, heterogeneous, and autonomous databases. *ACM Computing Surveys*, 22 (3), pp183-236.

Stal, M. 2002. Web Services: Beyond Component-based Computing. *Communications of the ACM*, 45 (10), pp71-76.

Sten, A. Davis, J. 2004. Extending the Web services model to IT services. *IN: Proceedings IEEE International Conference on Web Service*, 2004. pp824-825.

Szyperski, C. 2002. *Component Software: Beyond Object-Oriented Programming*. 2nd Edition. Addison-Wesley.

Thakkar, S. Ambite, L. Knoblock, A. 2005. Composing, optimizing, and executing plans for bioinformatics web services. *VLDB Journal*. 14 (3), pp33-353.

Thompson, H. Beech, D. Maloney, M. Mendelsohn, N. 2001. XML Schema Part 1: Structures. Available from : < http://www.w3.org/TR/xmlschema-1/>. [Accessed 20 August 2006].

Ullman J. 1997. Information Integration Using Logical Views. *IN: Proc. of the 6th Int. Conf. on Database Theory (ICDT-97), Lecture Notes in Computer Science*. Springer Verlag. pp19-40.

Van Deursen, A. Klint, P. Visser, J. 2000. Domain-specific languages: an annotated bibliography. *ACM SIGPLAN Notices*. 35 (6). New York: ACM Press, pp26-36.

Velegrakis, Y. Miller, J. Popa, L 2004. Preserving mapping consistency under schema changes. *VLDB Journal*. 13 (3), pp274-293.

Velegrakis, Y. Miller, R. Mylopoulos, J. 2005. Representing and Querying Data Transformations. *IN: Proceedings of the 21st International Conference on Data Engineering (ICDE'05)*. pp81-92.

Wallace, M. Runciman, C. 1999. Haskell and XML: Generic Combinators or Type-Based Translation? *ACM SIGPLAN Notices*, 34 (9), pp148-159.

Widom, J. 1995. Research problems in data warehousing. *IN: Proceedings of 4th International Conference on Information and Knowledge Management.* pp29-30.

Wiederhold, G. 1992. Mediators in the architecture of future information systems. *IEEE Computer*, 25 (3), pp38-49.

Willcocks, P. Lacify, C. 1998. The sourcing and outsourcing of IS: Shock of the New? *Strategic Sourcing of Information Technology: Perspective and Practices.* P. Willcocks and C. Lacity (ed)s. Chichester : Wiley. 1998.

Zamboulis, L. Poulovassilis, A. 2004. Using AutoMed for XML Data Transformation and Integration. *IN: Proceedings of DIWeb Workshop – CaiSE'04.* pp58-69.

Zoof, M.1977. Query By Example: A Data Base Language. *IBM Systems Journal.* 16 (4), pp324-343.

Zhu, F. Turner, M. Kotsiopoulos, I. Bennett, K. Russell, M. Budgen, D. Brereton, P. Keane, J. Layzell, P. Rigby, M. Xu, J. 2004. Dynamic Data Integration Using Web Services. *IN: 2nd International Conference on Web Services (ICWS)*, 2004. San Diego.

Zhu, Y. Pahl, C. 2006. Automating the Construction of Software Connectors for Adaptive Service Architectures. *Workshop on Applying Service Oriented Architectures to Adaptive Information Systems SOA-AIS* 2006.

Zhu, Y. Pahl. C. 2006. Mediated Data Integration and Transformation for Web service-based Software Architectures. *IN: Proceedings of European Conference on Web services ECOWS'2006* (Accepted as Poster paper).

Zhu, Y. Pahl, C. 2006. Data Integration through Service-based Mediation for Web-enabled Information Systems *IN*: D. Brandon (ed) *Software Engineering for Modern Web Applications*. Hershey : Idea Group Inc.

# Appendix A

## XML schema definitions for all Web services

In this appendix, we describe the XML schema definitions for both the global schema and the schemas for various Web service providers that are used in examples in this thesis. The XML schema language provides the defining framework for creating and validating XML documents by specifying the valid structure, constraints, and data types for the various elements and attributes of an XML document. We have demonstrated that the schema language plays a key role in specifying integration rules between the global schema and the local schemas. Business analysts design the global XML schema without knowing the schema definitions of Web service providers in Fig.A-1. We list the diagrams of the global schema in Fig.A-2 and Fig.A-3. In this thesis, we take the following three Web service providers to illustrate the problem setting and the solution. We also list the sample output returned from the mediator Web service after data integration and mediation in Fig. A-4.

**The "Customers" Web service provider**. The information source stores all the customer related information including the customer contact information and their service support identifier information.

**The "Services and systems" Web service provider.** The information source stores the information related to services that hosted by the organization "ERP Online" and the systems that support the services underneath.

**The "Service requests" Web service provider.** The provider provides information related to current and history service requests on the services and systems over the years with the service hosting company.

```
<xs:schema xmlns:xs=http://www.w3.org/2001/XMLSchema
elementFormDefault="qualified"
attributeFormDefault="unqualified">
      <xs:element name="CustomerArray"
      type="CustomerArrayType"/>
      <xs:complexType name="CustomerArrayType">
            <xs:sequence>
                  <xs:element name="Customer"
            type="CustomerType" maxOccurs="unbounded"/>
            </xs:sequence>
      </xs:complexType>
      <xs:complexType name="CustomerType">
```

```
<xs:sequence>
<xs:element name="customerName" type="xs:string"/>
<xs:element name="companyId" type="xs:string"/>
<xs:element name="supportIdentifier"
type="SupportIdentifierType"
maxOccurs="unbounded"/>
<xs:element name="Services" type="ServicesType"/>
</xs:sequence>
</xs:complexType>
<xs:complexType name="SupportIdentifierType">
      <xs:sequence>
      <xs:element name="CustomerSupportIdentifier"
      type="xs:string"/>
      <xs:element name="IsoCountryCode"
      type="xs:string"/>
      </xs:sequence>
</xs:complexType>
<xs:complexType name="ServicesType">
      <xs:sequence>
      <xs:element name="Service" type="ServiceType"
      maxOccurs="unbounded"/>
      </xs:sequence>
</xs:complexType>
<xs:complexType name="ServiceType">
      <xs:sequence>
      <xs:element name="serviceCustomerName"
      type="xs:string"/>
      <xs:element name="Systems"
      type="SystemsType"/>
      </xs:sequence>
</xs:complexType>
<xs:complexType name="SystemsType">
      <xs:sequence>
      <xs:element name="System" type="SystemType"
      maxOccurs="unbounded"/>
      </xs:sequence>
</xs:complexType>
<xs:complexType name="SystemType">
      <xs:sequence>
      <xs:element name="SystemIdentifier"
      type="xs:string"/>
      <xs:element name="Type" type="xs:string"/>
      <xs:element name="Status" type="xs:string"/>
      <xs:element name="Machines"
      type="MachinesType"/>
      </xs:sequence>
</xs:complexType>
<xs:complexType name="MachinesType">
      <xs:sequence>
      <xs:element name="Machine" type="MachineType"
      maxOccurs="unbounded"/>
      </xs:sequence>
</xs:complexType>
<xs:complexType name="MachineType">
      <xs:sequence>
```

```
            <xs:element name="HostName"
            type="xs:string"/>
            <xs:element name="ConfigurationTargets"
            type="ConfigurationTargetsType"/>
            <xs:element name="Status" type="xs:string"/>
            </xs:sequence>
      </xs:complexType>
      <xs:complexType name="ConfigurationTargetsType">
            <xs:sequence>
            <xs:element name="Target" type="TargetType"
            maxOccurs="unbounded"/>
            </xs:sequence>
      </xs:complexType>
      <xs:complexType name="TargetType">
            <xs:sequence>
            <xs:element name="SystemIdentifier"
            type="xs:string"/>
            <xs:element name="Type" type="xs:string"/>
            <xs:element name="Status" type="xs:string"/>
            </xs:sequence>
      </xs:complexType>
</xs:schema>
```

Fig.A-1 The XML schema definition of the virtual global schema.

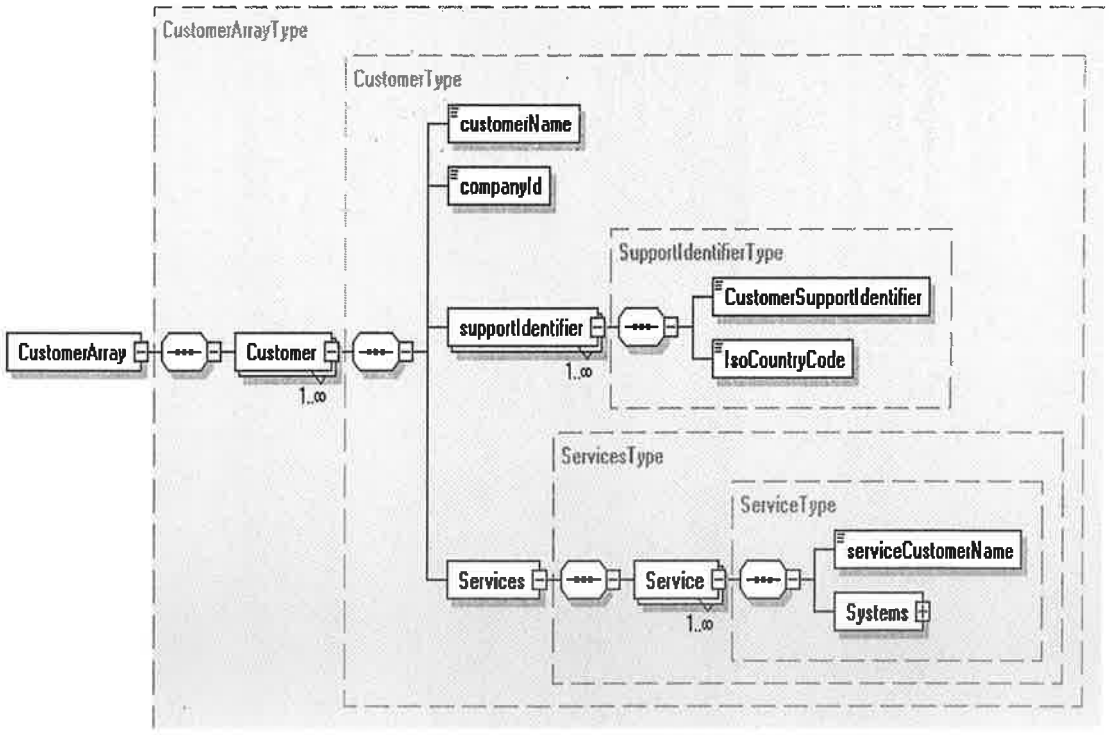


Fig.A-2 The XML schema diagram of the global schema.

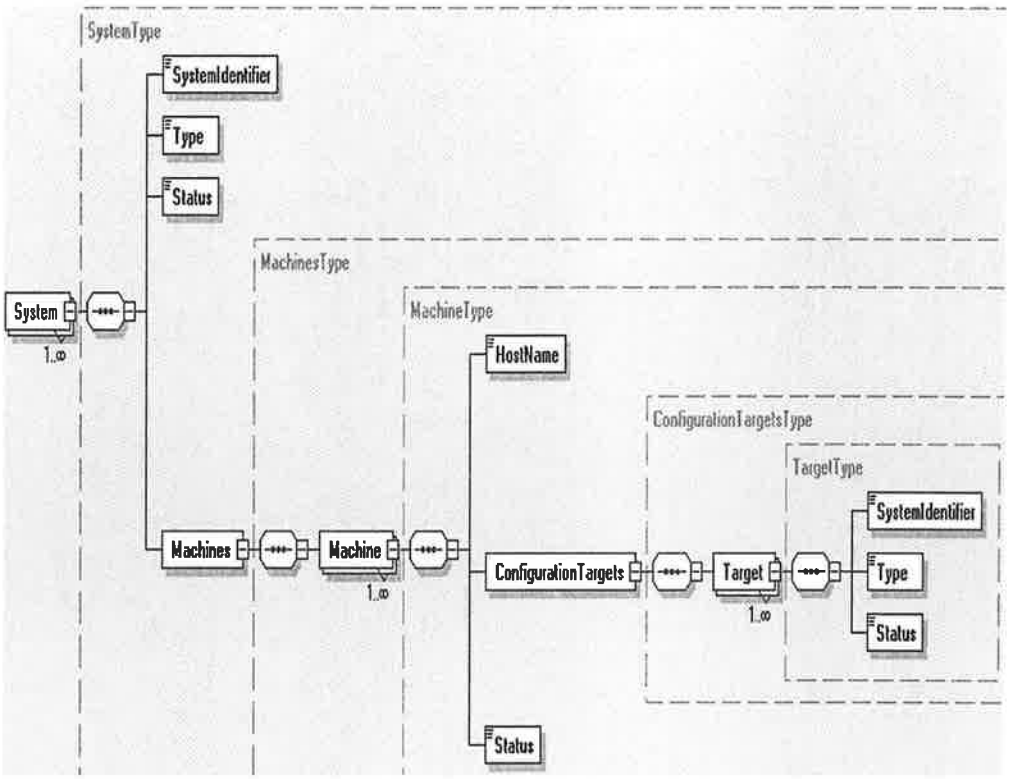


Fig.A-3 The system type diagram in the global schema.

We also list a sample of the instance of the global XML schema in Fig. A-4.

```
<CustomerArray>
<Customer>
    <nameAsContracted>Buy and Sale Online</nameAsContracted>
    <supportidentifier>
          <CustomerSupportIdentifier>840</CustomerSupportIden
          tifier>
          <ISOCountryCode>1398</ISOCountryCode>
    </supportidentifier>
    <supportidentifier>
          <CustomerSupportIdentifier>372</CustomerSupportIden
          tifier>
          <ISOCountryCode>1399</ISOCountryCode>
    </supportidentifier>
    <services>
          <serviceCustomerName>Technology</serviceCustomerNam
          e>
          <system>
    <SystemIdentifier>DBUY</SystemIdentifier>
    <Type>Test</Type>
    <Status>Active</Status>
    </system>
    </services>
    <service>
          <serviceCustomerName>Applicatioins</serviceCustomer
          Name>
    <system>
                <SystemIdentifier>TBUY</SystemIdentifier>
    <Type>Test</Type>
```

```
      <Status>Active</Status>
      </system>
      <system>
<SystemIdentifier>PBUY</SystemIdentifier>
      <Type>Production</Type>
      <Status>Active</Status>
      </system>
      </services>
      </Customer>
      </CustomerArray>
```

Fig. A-4 Sample output from the mediator Web service

# Appendix B

## Xcerpt query programs

We demonstrate the data mediation process in light of example. The example takes the XML data from the Web service provider "Services and systems" at runtime. This example is an immediate composite rule "Machines" that is used to provide the query to the data transformation goal described in Fig. B-1. The Xcerpt query in Fig. B-2 is a ground rule that populates the data term "raw machines" from the XML document at runtime.

```
CONSTRUCT
      Machines [
            all machine-of-system [
                  machine[
                        HostName [ var HostName ],
                        ConfigurationTargets [
                              target [
                                    Name [var TargetName],
                                    Status [var TargetStatus],
                                    Type[var TargetType]
                              ]
                        ]
                  ],
                  System [var ENV]
            ]
      ]
      FROM
            VAR raw_machines ->return [[
                  item [[
                        name [ var ENV ],
                        hostCollection [[
                        HostName [var HostName],
                        targetCollection[
                        item[
                              name [var TargetName],
                              status[var TargetStatus],
                              type[var TargetType]
                        ]
                  ]
            ]]
      ]]
]]
END
```

Fig. B-1 Xcerpt query program to transform the XML data from the Web service provider "Service and System".

```
CONSTRUCT
      Var raw-machines
FROM
      In resource ("<return>…</return>");
```

```
END
```

Fig. B-2 The ground rules to populate the data term "Machines" .

We list the Xcerpt query program in Fig. B-3 that is generated at runtime to connect the mediator Web service "Customer" to the dependent Web services providers such as the "Customers", the "Services and Systems", and the "Services requests". The immediate composite rule " " It integrates and mediates the XML data from these three Web service providers in the connector construction component. The transformed XML document is wrapped in SOAP messages and sent back the mediator Web service "Customer" at runtime.

```
CONSTRUCT
      CustomerArray [
            all   Customer[
                  var customerName,
                  all var Supportidentifier,
                  all services [
                        var ServiceName,
                        all system [
                              var SystemId,
                              var Type,
                              var Status,
                              all var Machine
                        ]
                  ]
            ]
      ]
FROM
      AND {
      CustomerArray [[
            Customer [[
                  var NameAsContracted ->
                  nameAsContracted[[ ]],
                  var ServiceOrganizationIdentifier ->
                  serviceOrganizationIdentifier[[ ]],
                  var Supportidentifier ->
                  supportidentifier[[ ]],
                  services [[
                        var ServiceCustomerName ->
                        serviceCustomerName[[ ]],
                        var GoLiveDate -> GoLiveDate[[ ]],
                        system [[
                              var SystemId ->
                        SystemIdentifier[[ var ENV ]],
                              var Type -> Type[[ ]],
                              var Status -> Status[[ ]]
                        ]]
                  ]]
            ]]
            },
            Machines [[
```

```
                machine-of-system [[
                    var Machine -> machine[[ ]],
                    System [ var ENV ]
                ]]
        ]]
    }
    END
    CONSTRUCT
    CustomerArray [
        ALL  Customer[
        nameAsContracted[var Name],
        companyId[var CompanyId],
        serviceOrganizationIdentifier[var OrgId],
        ALL supportidentifier[
            CustomerSupportIdentifier [var Code],
            ISOCountryCode [var CSI]
        ]
        ]
    ]
    FROM
    arrayOfCustomer[[
        item [[
            orgName[var Name],
            companyId[var CompanyId],
            gcdbOrgId [var OrgId],
            countryCode[var Code],
            csiNumber[var CSI]
        ]]
    ]]
    END
    CONSTRUCT
        ArrayOfCustomer [
            var arrayOfCustomer
        ]
    FROM
IN resource {"<arrayOfCustomer>…</arrayOfCustomer>"},
    arrayOfCustomer[[ var arrayOfCustomer ]]

    END
```

Fig. B-3 The Xcerpt query program to generate the instance of the global XML schema.

# Appendix C

## The metadata for the "Customer" mediator Web service

The "Customer" mediator Web service is the mediator in which the WS-BPEL integration process flows are responsible for orchestrating the Web service providers. At runtime, it is invoked from the query component in Section 5.3.2. Subsequently it invokes the connector construction component in order to integrate the XML documents according to the global XML schema. In this section, the metadata of the Web service including the WSDL definition is described in Fig. C-1. The partnerLink definition for the "Services and Systems" Web service provider is illustrated in Fig. C-2. The partnerLink definition for "Customer" Web service provider is outlined in Fig. C-3. The partnerLink definition for "Services requests" Web service provider is in Fig. C-4. The partnerLink definition for "The Service Connector Generator" Web service is in Fig. C-5. The partnerLink definition for "Query Component" Web service is in Fig. C-6.

```
<definitions name="CustomerMediatorWS"
targetNamespace="http://buyonline.com"
      xmlns:tns=" http://buyonline.com"
      xmlns:plnk="http://schemas.xmlsoap.org/ws/2003/05/partne
      r-link/"
      xmlns:xsd="http://www.w3.org/2001/XMLSchema"
      xmlns="http://schemas.xmlsoap.org/wsdl/">
      <types>
      <schema attributeFormDefault="qualified"
      elementFormDefault="qualified"
      targetNamespace="http://buyonline.com"
      xmlns="http://www.w3.org/2001/XMLSchema"
      xmlns:global="http://buyonline.com/types"
      xmlns:tns=" http://buyonline.com">
      <import namespace="http://buyonline.com/types"
      schemaLocation="
      http://buyonline.com/xmlschema/globalTypes.xsd"/>
            <message name="GetCustomerRequestMessage">
                  <part name="payload"
                  element="tns:GetCustomerRequest"/>
            </message>
            <message name="CustomerResponseMessage">
                     <part name="payload"
            element="tns:CustomerObject"/>
            </message>
            <portType name="Customer">
                     <operation name="getCustomer">
                        <input
                  message="tns:GetCustomerRequestMessage"/>
```

```
                    <output
                message="tns:CustomerResponseMessage"/>
                </operation>
            </portType>
            <plnk:partnerLinkType name="CustomerMediatorWS_PL">
                <plnk:role name="CustomerMediatorWS">
                        <plnk:portType name="tns:Customer"/>
                    </plnk:role>
            </plnk:partnerLinkType>
            <service name="CustomerMediatorWS" />
</definitions>
```

Fig. C-1 The WSDL definition of the mediator Web service "Customer".

```
<definitions  name="ServicesAndSystemsWSProvider_PL"
targetNamespace="http://www.buyonline.com/ws/ServicesAndSystem
sWS.wsdl"
      xmlns:tns="
      http://www.buyonline.com/ws/ServicesAndSystemsWS.wsdl"
      xmlns="http://schemas.xmlsoap.org/wsdl/"
      xmlns:plnk="http://schemas.xmlsoap.org/ws/2003/05/partne
      r-link/">
            <import location="
            http://www.buyonline.com/ws/ServicesAndSystemsWS?WS
            DL"/>
            <plnk:partnerLinkType
            name="ServicesAndSystemsWSPortTypeLink">
            <plnk:role name="ServicesAndSystemsWSProvider">
            <plnk:portType name="tns:
            ServicesAndSystemsWSPortType"/>
            </plnk:role>
      </plnk:partnerLinkType>
</definitions>
```

Fig. C-2 The partnerLink definition for "Services and systems" Web service provider.

```
<definitions  name="CustomersWSProvider_PL"
      targetNamespace="http://www.buyonline.com/ws/ServiceWS.w
      sdl"
      xmlns:tns="
      http://www.buyonline.com/ws/CustomersWS.wsdl"
      xmlns="http://schemas.xmlsoap.org/wsdl/"
      xmlns:plnk="http://schemas.xmlsoap.org/ws/2003/05/partne
      r-link/">
      <import location="
      http://www.buyonline.com/ws/CustomersWS?WSDL"/>
      <plnk:partnerLinkType name="CustomersWSPortTypeLink">
            <plnk:role name="CustomersWSProvider">
                  <plnk:portType name="tns:
            CustomersWSPortType" />
            </plnk:role>
      </plnk:partnerLinkType>
</definitions>
```

Fig. C-3 The partnerLink definition for "Customer" Web service provider.

```
<definitions  name="SRWSProvider_PL"
      targetNamespace="http://www.buyonline.com/ws/SRWS.wsdl"
      xmlns:tns=" http://www.buyonline.com/ws/SRWS.wsdl"
      xmlns="http://schemas.xmlsoap.org/wsdl/"
      xmlns:plnk="http://schemas.xmlsoap.org/ws/2003/05/partne
      r-link/">
      <import location="
      http://www.buyonline.com/ws/SRWS?WSDL"/>
      <plnk:partnerLinkType name="SRWSPortTypeLink">
            <plnk:role name="SRWSProvider">
                  <plnk:portType name="tns:SRWSPortType" />
            </plnk:role>
      </plnk:partnerLinkType>
</definitions>
```

Fig. C-4 The partnerLink definition for "Service requests" Web service provider.

```
<definitions  name="ServicesConnectorGenerator_PL"
      targetNamespace="http://www.buyonline.com/ws/SCGWS.wsdl"
      xmlns:tns=" http://www.buyonline.com/ws/SCGWS.wsdl"
      xmlns="http://schemas.xmlsoap.org/wsdl/"
      xmlns:plnk="http://schemas.xmlsoap.org/ws/2003/05/partne
      r-link/">
      <import location="
      http://www.buyonline.com/ws/SCGWS?WSDL"/>
      <plnk:partnerLinkType name="SCGWSPortTypeLink">
            <plnk:role name="ServicesConnectorGenerator">
                  <plnk:portType name="tns:SCGWSPortType" />
            </plnk:role>
      </plnk:partnerLinkType>
</definitions>
```

Fig. C-5 The partnerLink definition for the "ServicesConnectorGenerator" Web service.

```
<definitions  name="QueryComponentWS_PL"
      targetNamespace="http://www.buyonline.com/ws/SCGWS.wsdl"
      xmlns:tns=" http://www.buyonline.com/ws/SCGWS.wsdl"
      xmlns="http://schemas.xmlsoap.org/wsdl/"
      xmlns:plnk="http://schemas.xmlsoap.org/ws/2003/05/partne
      r-link/">
      <import location="
      http://www.buyonline.com/ws/SCGWS?WSDL"/>
      <plnk:partnerLinkType
      name="QueryComponentWSPortTypeLink">
            <plnk:role name="QueryComponentWS">
                  <plnk:portType
            name="tns:QueryComponentWSPortType" />
            </plnk:role>
      </plnk:partnerLinkType>
</definitions>
```

Fig. C-6 The partnerLink definition for the "Query Component" Web service.