Constraints and Triggers to Enhance XML-based Data Integration Systems

Von der Fakultät Informatik, Elektrotechnik und Informationstechnik
der Universität Stuttgart
zur Erlangung der Würde eines
Doktors der Naturwissenschaften (Dr. rer. nat.)
genehmigte Abhandlung

Vorgelegt von

Jing Lu

aus Jiangsu, China

Hauptberichter:	Prof. Dr.-Ing. habil. Bernhard Mitschang
Mitberichter:	Prof. Dr.-Ing. Stefan Deßloch

Tag der mündlichen Prüfung: 27. 04. 2009

Institut für Parallele und Verteilte Systeme (IPVS)
der Universität Stuttgart
Bundesrepublik Deutschland
2009

Acknowledgements

"A teacher for a day is a father for a lifetime." This is an old Chinese saying. It means that one should appreciate and respect those people who once taught him something. It is not an accident that in German the supervisor of a PhD candidate is called "Doktorvater". So first, I thank my Doktorvater, Prof. Dr. -Ing. habil. Bernhard Mitschang, who encouraged me to work on this topic, for his continuous support, bright ideas, patience and wise suggestions. Furthermore, he gave advice on how to be a good paper writer and how to be a good presenter. I will cherish this appreciation all my life.

Second, I would like to thank Prof. Dr.-Ing. Stefan Deßloch, for spending his valuable time on reviewing this work and giving me many significant feedbacks.

Third, I must thank my parent and my husband, for their love, understanding and tolerance. And I also appreciate years of support and encouragement from Prof. Qian Wang and the family of Jianshe Zhu.

I owe many thanks to Sylvia Radeschütz for her modification and advice of the German abstract and to Lu Li for her careful proof-reading of this work. The add-oils from Zhao Chen, Chunmei Yin, Lianfeng Yang, Jianfeng Zhang and Hao Wang always stimulated me to persevere.

"No pains, no gains." I would like to share this adage with my son, Xinhan Tao, who gave me the most motivation to finish this work and who gave me the most sunshine, the smile, and the happiness during my study in Stuttgart.

Stuttgart, April 27, 2009. *Jing Lu*

Abstract

XML is becoming one of the main technological integredients of the Internet. It is now accepted as the standard for information exchange. XML-based data integration system, which enables sharing and cooperation with legacy data sources, arises as a more and more important data service provider on the web. These services can provide the users with a uniform interface to a multitude of data sources such as relational databases, XML files, text files, delimited files, Excel files, etc. Users can thus focus on what they want, rather than think about how to obtain the answers. Therefore, users do not have to carry on the tedious tasks such as finding the relevant data sources, interacting with each data source in isolation using the local interface and combining data from multiple data sources.

Users are always expecting better query performance and data consistency from the data integration systems. This work proposes an approach to support constraints and triggers in the XML-based data integration system in order to optimize queries and to enforce data consistency. Constraints and triggers have long been recognized to be useful in semantic query optimization and data consistency enforcement in relational databases. This work first gives an approach to use constraints from the heterogeneous data sources to semantically optimize queries submitted to the XML-based data integration system. Different constraints from the data sources are first integrated into a uniform constraint model. Then the constraints in the uniform constraint model are stored in the constraint repository. Traditional semantic query optimization techniques in the relational database are analyzed and three of them are reused and applied by the semantic query optimizer for XML-based data integration system. Among them are detection of empty results, join elimination and predicate elimination. Performance is analyzed according to the data source type and the data volume. The semantic query optimizer works best when the data sources are non-relational, the data volume is huge and the execution cost is expected to be high.

In order to make the XML-based data integration system fully equipped with data manipulation capabilities, programming frameworks which support update at the integration level are being developed. This work discusses how to realize update in the XML-based data integration system under the Service Data Objects programming framework. When the user is permitted to submit updates, it is necessary to guarantee data integrity and enforce active business logics in the data integration system. This work presents an approach by which active rules including integrity constraints are enforced by XQuery triggers. An XQuery trigger model in conformance to XQuery update model proposed by W3C is defined. How to define active rules and integrity constraints by XQuery triggers is discussed. Triggers and constraints are stored in the trigger repository. The architecture supporting XQuery trigger service in the XML-based data integration system is proposed. Important components including event detection, trigger scheduling, condition evaluation, action firing and trigger termination are discussed. The whole XQuery trigger service architecture above a data integration system is implemented in BEA AquaLogic DataService Platform under the Service Data Objects programming framework. Experiments show active rules and integrity constraints are enforced easily, efficiently and conveniently at the global level.

Constraints and triggers play an important role in XML-based data integration systems. Using constraints and triggers in the XML-based data integration system we can efficiently improve query performance and enforce data consistency.

Zusammenfassung

Web Services sind eine neue Technologie für Web-Anwendungen. Doch werden ein großer Teil der Daten weiterhin in Legacy-Datenquellen gespeichert, wie z.B. relationale und objekt- orientierte Datenbanken, Textdateien, XML-Dateien, HTML-Dateien und Excel-Dateien. Informationsintegration ist seit langem ein zentrales Forschungsthema im Bereich der Datenbanken und der Künstlichen Intelligenz. Die älteren Datenbanksysteme waren meisterns noch Einzelsysteme (standalone). Mittlerweile ist der große Nutzen aus der Integration von Informationen aus verschiedenen Quellen bekannt, und vielfach auch gewünscht bzw. benötigt.

Die Nachrichten, die mit Web Services ausgetauscht werden, basieren hauptsächlich auf XML. XML hat sich als dominierender Standard für den Austausch von Informationen über das Internet etabliert. Als Anfragesprache für XML dient XQuery, das sowohl sehr leistungsfähig als auch leicht bedienbar und erlernbar ist. In den letzten Jahren sind viele XML basierte Datenintegrationsssysteme auf den Markt gekommen. Sie bieten eine einheitliche Schnittstelle für die Benutzer an, Anfragen an heterogene Datenquellen zu stellen. Die Anwender können sich somit darauf konzentrieren, was sie an Information erfahren wollen, anstatt darauf, wie sie die Information beziehen können. So muss sich der Benutzer nicht mit der langwierigen Aufgabe beschäftigen, die relevanten Datenquellen zu finden, mit jeder Datenquelle über die jeweilige lokale Schnittstelle separat zu interagieren und die Ergebnisse am Ende wieder zu kombinieren. In einem XML-basierten Datenintegrationssystem gibt es für jede Datenquelle einen Wrapper. Dieser Wrapper führt ein XML-Schema aus, das den Inhalt der entsprechenden Datenquelle in XML beschreibt. Der Anfrageprozessor des XML-basierten Datenintegrationssystems nimmt eine Anfrage in XQuery von den Benutzern oder Anwendungen entgegen, analysiert die Anfrage, zerlegt sie in einzelne Teilanfragen und übergibt sie dem jeweiligen Wrapper. Die Wrapper übersetzen die Anfragen in die lokale Anfragesprache und übersetzen nach ihrer Ausführung die Teilergebnisse zurück in XML.

Anwender erwarten eine immer bessere Anfrageperformanz und Datenkonsistenz von den XML-basierten Datenintegrationssystemen. Diese Arbeit stellt ein Konzept vor, wie Integritätsbedingungen und Trigger in XML-basierten Datenintegrationssystemen unterstützt werden können, um Anfragen zu optimieren und die Datenkonsistenz zu garantieren. Integritätsbedingungen und Trigger werden in relationalen Datenbanken genutzt, um semantische Anfrageoptimierungen durchzuführen und Datenkonsistenz zu sichern.

Das größte Problem bei der Optimierung von Anfragen in einem XML-basierten Datenintegrationssystem ist, dass die Datenkommunikation bei dem Datenaustausch zwischen Benutzer, Datenintegrationssystem, Wrapper und Datenquellen sehr aufwändig und teuer ist. In dieser Situation ist eine Optimierung durch semantische Regeln attraktiv, um die Kosten der Kommunikation zu reduzieren. Die Optimierung durch semantische Regeln ist bekannt unter dem Namen "Semantic Query Optimization" (SQO). Zu diesem Thema sind viele Forschungsarbeiten im Bereich der relationalen und deduktiven Datenbanken erschienen. SQO nutzt die Integritätsbedingungen zur Verbesserung der Effizienz der Anfrageausführung. Mit SQO kann mehr Effizienz erreicht werden als nur durch syntaktische Anfrageoptimierungstechniken allein, weil ein syntaktischer Optimierer semantisches Wissen nicht verstehen kann und damit zu einer suboptimalen Ausführung für die jeweilige Suchanfrage führt. Deshalb können diejenigen Anfragen, die eigentlich ohne Zugriff auf die Datenquelle beantwortet werden könnten, nicht von dem syntaktischen Optimierer erkannt werden. Ein syntaktischer Optimierer kann keine semantisch redundanten Bedingungen erkennen und löschen oder zusätzliche nützliche Bedingungen der Anfrage hinzufügen, um die Gesamtkosten für die Anfrageausführung minimal zu halten.

Diese Arbeit stellt zunächst einen Ansatz zur Verwendung der Integritätsbedingungen von heterogenen Datenquellen vor, um Anfragen an XML-basierte Datenintegrationssysteme zu optimieren. Verschiedene Integritätsbedingungen aus heterogenen Datenquellen werden zuerst in ein einheitliches Modell (Uniform Constraint Model) integriert. Dann werden die Integritätsbedingungen aus dem Uniform Constraint Model in das Repositorium für Integritätsbedingungen gespeichert. Traditionelle semantische Anfrageoptimierungstechniken der relationalen Datenbanken werden analysiert, drei davon ausgewählt und für die XML-basierten Datenintegrationssysteme implementiert: die Erkennung von leeren Anfrageergebnissen, das Löschen von Joins und das Löschen von Prädikaten. Die Anfrageperformanz wird analysiert, sowie sortiert nach Datenquelle und Datenvolumen. Der semantische Anfrageoptimierer erreicht die beste Leistung, wenn die Datenquellen nicht-relational sind, das Datenvolumen groß ist und die Ausführungskosten hoch sind.

Um XML-basierten Datenintegrationssystemen eine vollständige Verwendung von Datenmanipulationen zu erlauben, wurde ein Framework entwickelt, das dem Benutzer die Erstellung von Updates auf der Integrationsebene ermöglicht. Wenn der Benutzer Updates durchführen darf, ist es notwendig, Datenintegrität und Datenkonsistenz in dem Datenintegrationssystem zu gewährleisten.

In den Datenintegrationssystemen liegen die Regeln zur Einhaltung der Datenkonsistenz in verschiedenen Formen vor, z.B. in verschiedenen Komponenten oder sogar im Programmcode. Deshalb ist die Wartung sehr schwer. So ist es eine herausfordernde Aufgabe, eine einheitliche Definition, Verwaltung und Wartung der Datenkonsistenzregeln zu entwerfen. Diese Anforderungen lassen sich am besten mit Triggern umsetzen. Trigger ermöglichen eine einheitliche und kompakte Beschreibung der aktiven Regeln und Integritätsbedingungen. Trigger haben eine einfache Syntax und können automatisch als Reaktion auf Ereignisse durchgeführt werden.

Diese Arbeit präsentiert eine Methode, in der Datenkonsistenzregeln durch XQuery-Trigger umgesetzt werden können. Es wird ein XQuery-Trigger-Modell definiert in Übereinstimmung zu dem XQuery-Update Standard von W3C. Wie Datenkonsistenzregeln dann mit den XQuery-Triggern definiert und in einem Triggerrepositorium gespeichert werden, wird anschließend diskutiert. Die gesamte Architektur wird präsentiert, einschließlich Ereigniserkennung, Auswertung der Bedingungen, Absetzen der Aktionen und dem Beenden des Triggers. Der Prototyp des XQuery-Trigger-Services ist auf BEA

Aqualogic DataService Platform implementiert. Die Evaluierungen zeigen, dass Datenkonsistenzregeln leicht und effizient auf globaler Ebene umgesetzt werden können.

Integritätsbedingungen und Trigger spielen eine wichtige Rolle bei XML-basierten Datenintegrationssystemen. Die Verwendung von Integritätsbedingungen und Triggern in XML-basierten Datenintegrationssystemen kann ihre Anfrageperformanz verbessern und ihre Datenkonsistenz sichern helfen.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

Web services are emerging as a new paradigm to build web applications. However, a large fraction of data continues to be stored in legacy data sources including relational databases, object-oriented databases, text files, XML files, delimited files, HTML files, Excel files, and so on. Information integration has long been recognized as a central problem of modern database systems. While early databases were self-contained, it is now generally realized that there is great value in taking information from various sources and making them work together as a whole [Wie92] [Ull97]. The goal of a data integration system is to provide a *uniform* interface to a multitude of data sources. This problem has received considerable attention from researchers in the fields of Artificial Intelligence and Database Systems. The most important advantage of a data integration system is that it enables users to focus on specifying *what* they want, rather than thinking *how* to obtain the answers. As a result, it frees the users from the tedious tasks of finding the relevant data sources, interacting with each source in isolation using a particular interface, and combining data from multiple sources [Hal01].

XML is one of the main technological ingredients in web services. It has emerged as a dominant standard for information exchange on the Internet. XQuery is a powerful and convenient language designed for querying XML. Nowadays there are many XML-based Data Integration Systems, where each wrapper exports an XML Schema describing the content of the corresponding source as XML. It also obtains and exports metadata describing the capabilities of the sources. The Query Processor of the XML-based data integration system accepts XQuery from the users or end-applications, parses it, decomposes it, pushes down the query plan and transforms it to the wrappers. The wrappers translate the queries into the local language and transform the local results into XML. Typical XML-based data integration systems include EXIP [PV02], Xyleme [ASV06], Silkroute [FTS00], BEA AquaLogic [Car06], XCalia Intermediation Core [Xca08], etc.

1.1.1 Improving Query Capabilities

Nowadays people are expecting more efficient queries from data integration systems where query optimization is greatly different from that in the traditional database context. First, since the sources are autonomous, the optimizer may have no statistics about the sources, or just unreliable ones. Hence, the optimizer cannot compare different plans, because their costs cannot be sufficiently well-estimated. Second, since the sources may have different processing capabilities, the query optimizer needs to consider the possibility of exploiting the query processing capabilities of a data source. Finally, in a traditional system, the optimizer can reliably estimate the time to transfer data from the disc to main memory. But in a data integration system, data is often transferred over a wide-area network, and hence delays may occur for a multitude of reasons. Therefore, even a plan that appears to be the best based on cost estimates may turn out to be inefficient if there are unexpected delays in transferring data from one of the sources accessed early on the plan.

The heterogeneity and web-orientation of modern applications have benefited from the flexibility of XML. A major difficulty in optimizing queries in an XML-based data integration system is that once a query is submitted, control over its execution becomes no longer possible [OB04]. Under this situation, using semantic rules to minimize the cost of communication becomes more attractive. Semantic query optimization (SQO) has been applied to relational and deductive databases [SO89]. SQO uses the integrity constraints associated with a database to improve the efficiency of query evaluation. With SQO, more efficiency can be achieved than by only using syntactic query optimization techniques, because a syntactic optimizer does not understand semantic knowledge, and thus leads to a suboptimal execution for the given query. Certain queries, which can be answered without any table scan in the database, cannot be detected by a syntactic optimizer, thus resulting in unnecessary database accesses. Furthermore, a syntactic optimizer cannot detect and eliminate semantically redundant restrictions or joins in the queries or introduce some useful restrictions and joins into a query to reduce the overall cost of query execution, either [CGK$^+$99]. SQO can also be adapted to XML-based data integration system using constraints from local data sources. An SQO optimizer for XML-based data integration systems uses the integrity constraints associated with the data sources of the data integration system to semantically optimize the queries submitted to the data integration system. When we want to develop an SQO optimizer for XML-based data integration systems, some problems arise naturally. First, constraints from data sources are normally heterogeneous. For example, there are relational constraints (CHECK, ASSERTION, etc), XML constraints, STEP/EXPRESS constraints, and so forth. In order to make the optimizer of the integration system understand the heterogeneous constraints, a uniform constraint model is needed. Second, there must be a wrapper to translate the heterogeneous constraints of the data sources to the uniform constraint model. Third, there must be a repository to store the constraints. Fourth, there must be an analysis to see which traditional SQO techniques fit for the XML-based data integration systems and which not. Details will be shown in Chapter 3.

1.1.2 Scaling up Data Consistency

Currently most of the XML-based data integration systems do not support ad-hoc XQuery update at the integration level because global update is inherently a difficult problem and until now W3C has not published the final standard of the XQuery Update facility. Programming frameworks are designed to solve this problem. We realize the global update in the XML-based data integration system by the Service Data Objects [BBNP03] framework. Details are shown in Chapter 4.

1.1.2.1 Ensuring Global Integrity Constraints

When the data integration system supports global update, data integrity constraints should be enforced. Integrity constraints specify those states of the data integration system that are considered to be semantically valid. In a data integration environment, integrity constraints might specify that replicated information is not contradictory, that information is not duplicated, that certain referential integrity constraints hold, or that some other condition is true over the data in multiple data sources. Although a lot of work has been done on transaction management in multidatabase systems, the treatment of semantic integrity constraints (e.g. uniqueness conditions), especially detecting potential violations and specifying necessary reactions in XML-based data integration systems, is often omitted [TC97].

Currently, constraints in the data integration systems are either not monitored at all, or monitored using ad-hoc techniques. Such techniques are error-prone and can lead to irreparable inconsistencies in the data sources of the data integration systems [CGMW96]. One could just ignore the problem, let all constraints be checked at the local level. This is unsatisfactory for the following reasons [GETE89]:

- Users of the data integration system would not get a clear conceptual picture of the constraints in the system. They often want to know why a global update has been rejected.
- In some cases the same constraints could be defined at all data sources. It is more efficient to check it once at the global level, rather than check it multiple times at multiple local databases.
- In some cases there are inherently global constraints, which are not derivable from local constraints. It is necessary to identify and provide a mechanism for enforcing them.
- There are some constraints, in which the check of one constraint needs the evaluation of the conditions on other data sources.

- There are some data sources, which do not have a constraint checking mechanism. The data integration system should give a compensation to ensure the constraints from these data sources.

In a traditional centralized or tightly-coupled distributed databases, transactions form the cornerstone of integrity constraint checking: before a transaction commits, it ensures that all integrity constraints are valid. If a constraint is violated, then the transaction may be aborted, the constraint may be corrected automatically or an error condition may be raised. While the facilities like atomic transactions, locking, and consistent queries are reasonable assumptions in centralized or tightly-coupled distributed environments, they typically do not hold in loosely-coupled heterogeneous environments. A data integration system is comprised of multiple heterogeneous data sources, including relational databases, object-oriented databases, text files, XML files, XML native databases, excel files, or even web applications. These data sources are integrated together through the data integration system to provide a global view to the user. Another characteristic of data integration system is that different data sources offer different facilities for constraint management, which makes constraint management more difficult. The lack of inter-site transaction mechanisms in data integration system renders traditional constraint checking mechanisms inapplicable. Therefore, in order to ensure the global data integrity in the data integration system, we need new constraint checking mechanisms. We first need a specification language to assign global integrity constraints in the data integration systems. Then we need new mechanisms to detect the update operation, to evaluate integrity condition and to avoid their violations. These requirements can be best satisfied by triggers. Details will be presented in Chapter 5.

1.1.2.2 Enforcing Active Rules in XML-DIS

Triggers, or active rules, are already well studied in the active database systems. Active database systems allow users to create rules to specify data manipulation operations to be executed automatically whenever certain events occur or conditions are met [Wid96]. Active database rules provide a general and powerful mechanisms for traditional database features such as integrity constraint enforcement, view maintenance, and authorization checking. Active database rules also support non-traditional database features such as version management, and workflow control.

When the data integration system supports global updates, data consistency should be enforced, e.g., when the user updates data source A, B might also be updated to keep data consistency between A and B. Currently, in the data integration systems data consistency enforcement rules are described in different forms, managed by different components or even embedded in the programming code whereby the maintenance becomes very hard. Therefore, it is demanding to define a uniform definition, management and maintenance of data consistency enforcement rules. These requirements can be best fulfilled by triggers [PD99] [VSCR00]. Triggers enable a uniform and compact description of active rules and integrity constraints, which are the foundation of data consistency, and facilitate the maintenance of them [SD95]. Triggers have a simple syntax and are automatically invoked in response to events. The simple execution model of triggers makes them a promising means for scaling up data consistency in data integration system. Triggers are processed in a specific environment defined as a data integration system service. In such environments, possible uses of triggers are numerous, for instance, view maintenance, global schema updates, verification and validation of global integrity constraints, notification, application and component integration and cooperation, etc [VSCR00].

Supporting trigger service in data integration system applications implies at least four challenges:

1. The inherent heterogeneity of the data integration system requires a flexible and simple trigger specification language.
2. The inherent heterogeneity of the system imposes the need for a flexible rule execution model adaptable to the characteristics of the participating data sources.
3. The active mechanism must deal with the autonomy of both the global system itself and the participating data sources. In a data integration system, data sources can keep their communication and execution autonomy.

Thus, they may share or not control information and they may continue to control their execution autonomy at any moment, independently of the integration. They can commit or abort local transactions at any time and this can affect the execution of global operations.

4. The events stemming from different contexts must be managed by the trigger service. Communication protocols are needed to observe events from their sources (data sources) and signal them to consumers (rules). Events are messages containing information about the integration and its component data sources. Therefore, they should be processed respecting information consistency, legacy and performance needs.

Details are shown in Chapter 5.

1.2 Problem Overview

In general, in order to support constraints and triggers in XML-based data integration systems, the following problems must be settled:

1. It is necessary to define a uniform constraint model to express the constraints coming from the local data sources.
2. The design of a constraint wrapper which can execute the translation between the local constraints and the uniform constraint model.
3. Analysis of the current SQO techniques to see which can be reused in the XML-based data integration system scenarios.
4. Implement the SQO techniques.
5. Performance analysis. Ensure that SQO really improves the query capability.
6. It is necessary to define the global constraint and trigger model for defining the data consistency enforcement rules.
7. The global trigger execution model should be defined and implemented including how to detect the event, evaluate the condition, and fire the actions.

This dissertation concentrates on settling the above-mentioned problems.

1.3 Contributions

Our contributions include:

1. We provide a uniform constraint model to represent constraints from different and typically heterogeneous data sources.
2. We introduce the Constraint Wrapper concept and present the algorithm to translate local constraints into the uniform constraint model automatically.
3. We establish the Constraint Repository, where constraints from local data sources are stored and maintained.
4. We provide the architecture of the semantic query optimizer and show how queries to different data integration systems are optimized by the optimizer. We describe the details of the query adapter, which is the kernel of the optimizer. We implement three SQO techniques of the optimizer: detection of empty results, join elimination and predicate elimination. We carry out experiments and analyze the performance.
5. We realize global update under the SDO programming framework in the XML-based data integration system.
6. We introduce the XQuery trigger specification language for the developer to assign data consistency enforcement rules including global integrity constraints over the data integration system.
7. We build up the architecture of data consistency rule enforcement in the data integration system and show how to detect events, evaluate conditions, and execute actions.
8. When there is an update from the application to the data integration system, we enforce constraints in order to ensure the data integrity in the XML-based data integration systems.
9. We implement the whole prototype and compare it with the traditional methods by implementing data consistency enforcement rules in the programming code.

1.4 Structure of this Work

Chapter 2 gives the fundamental concepts and techniques underlying of this work. This chapter includes the general introduction of data integration systems, different global data models, different integration models, etc; XML and XQuery; XQuery Update; constraints and triggers; and so on.

Chapter 3 first proposes the constraint model in the global level to express the local constraints. Then it introduces how to establish the constraint reporitory and the methodology to develop a constraint wrapper. This chapter discusses also how to use constraints associated to the data sources as query rewriting rules to optimize queries semantically in a data integration system. Experiments and evaluation are shown at the end of that chapter.

Chapter 4 first introduces the well-known and difficult problem of updating a view. Then it introduces the XQuery Update Facility 1.0 by W3C. The state-of-art of realizing XQuery updates in the XML-based data integration system is proposed. How to realize update through Service Data Objects follows. We give details on the implementation of realizing updates in the XML-based data integration system under the SDO programming framework.

Chapter 5 proposes the XQuery trigger model and shows how to define active rules and integrity constraints using this model. It also discusses architectural issues and the details of enforcing integrity constraints and active rules in a data integration system. Main components such as event detection, trigger scheduling, condition evaluation, action firing, trigger termination and failure handling are discussed. An implementation view and an evaluation are given, too.

Chapter 6 covers the related work. Our work is compared with the related work in three aspects: semantic query optimization, data integrity and active system.

Finally, chapter 7 gives the conclusion of this work.

Chapter 2

Fundamentals

This chapter will cover the fundamental knowledge of this work. We will begin with the definition of a data integration system. Then we will discuss the integration model. After that relational data integration systems and XML-based data integration systems will be compared. Then constraints and triggers will be introduced both in relational databases and in XML databases. We will discuss the semantic query optimization and the active database systems at the end.

2.1 XML-based Data Integration System

2.1.1 Data Integration System

2.1.1.1 Data Mediation

Information integration has long been recognized as a major issue in modern database systems. The main characteristic distinguishing data integration systems from distributed and parallel database systems is that the data sources underlying the system are *autonomous*. In particular, a data integration system provides access to *pre-existing* sources, which were created independently. Unlike multidatabase systems, a data integration system must deal with a large and constantly changing set of data sources. These characteristics raise the need for richer mechanisms for describing the data. In particular, a data integration system requires a flexible mechanism for describing contents of sources that may have overlapping contents, which are described by complex constraints, and which may be incomplete or only partially complete.

A common integration architecture is shown in Figure 2.1 [Wie92] [Ull97].

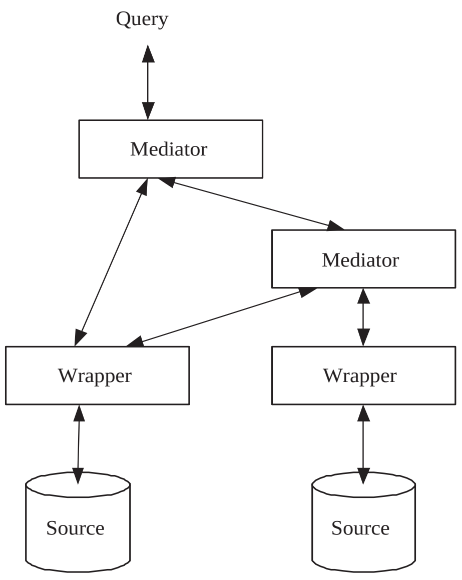


Fig. 2.1 Architecture of a Common Integration Architecture

Several sources are *wrapped* by software that translates between the source's local language, model, and concepts and the global concepts shared by some or all of the sources. System components, here called *mediators* [Wie92], obtain information from one or more components below them, which may be wrapped sources or other mediators. Mediators also provide information to components above them and to external users of the system. In a sense, a mediator is a view of the data found in one or more sources. Data does not exist at the mediator, but one may query the mediator as if data was stored there. It is the responsibility of the mediator to mediate with its sources or other wrappers to find the answer to the query.

Figure 2.2 shows a detailed architecture of a data integration system.

The wrappers are responsible to generate a view of the local data sources and the views are stored in the data integration systems. Different from a data warehouse, the views are not materialized but virtual.

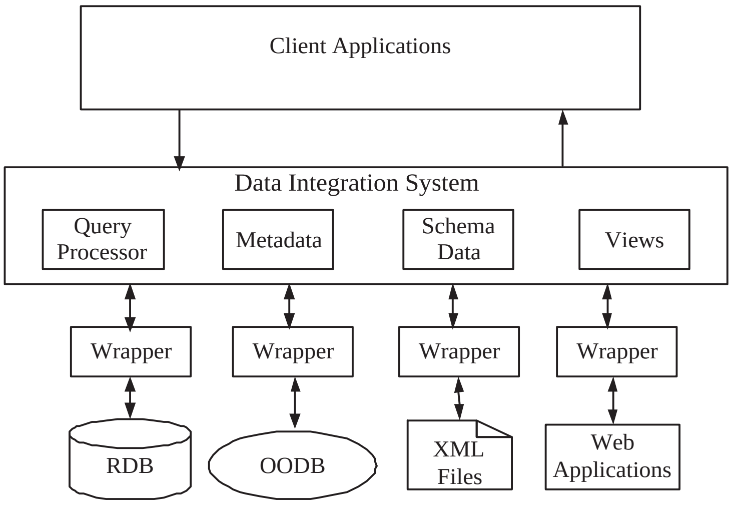


Fig. 2.2 Detailed Architecture of a Data Integration System

2.1.1.2 A Theoretical Perspective of Data Integration

The most important ideas in a data integration system are the global schema, the sources, and the mapping between them. [Len02] formalizes a data integration system $\mathcal{I}$ as a triple $< \mathcal{G}, \mathcal{S}, \mathcal{M} >$, where

- $\mathcal{G}$ is the *global schema*, expressed in a language $\mathcal{L}_{\mathcal{G}}$ over an alphabet $\mathcal{A}_{\mathcal{G}}$. The alphabet comprises a symbol for each element of $\mathcal{G}$ (i.e., a relation if $\mathcal{G}$ is relational, a class if $\mathcal{G}$ is object-oriented, etc.)
- $\mathcal{S}$ is the *source schema*, expressed in a language $\mathcal{L}_{\mathcal{S}}$ over an alphabet $\mathcal{A}_{\mathcal{S}}$. The alphabet $\mathcal{A}_{\mathcal{S}}$ includes a symbol for each element of the source schema.
- $\mathcal{M}$ is the *mapping* between $\mathcal{G}$ and S, constituted by a set of *assertions* of the form:

  $q_{\mathcal{S}} \rightarrow q_{\mathcal{G}}$,

  $q_{\mathcal{G}} \rightarrow q_{\mathcal{S}}$

  where $q_{\mathcal{S}}$ and $q_{\mathcal{G}}$ are two queries of the same form, over the source schema $\mathcal{S}$, and over the global schema $\mathcal{G}$ respectively. Queries $q_{\mathcal{S}}$ are expressed in a query language $\mathcal{L}_{\mathcal{M},\mathcal{S}}$ over the alphabet $\mathcal{A}_{\mathcal{S}}$. Queries $q_{\mathcal{G}}$ are expressed in a query language $\mathcal{L}_{\mathcal{M},\mathcal{G}}$ over the alphabet $\mathcal{A}_{\mathcal{G}}$. Intuitively, an assertion $q_{\mathcal{S}} \rightarrow q_{\mathcal{G}}$ specifies that the concept represented by the query $q_{\mathcal{S}}$ over the sources corresponds to the concept in the global schema represented by the query $q_{\mathcal{G}}$. The explanation of an assertion of type $q_{\mathcal{G}} \rightarrow q_{\mathcal{S}}$ is similar.

In the following sections, we assume that both the mediated schema and the source schema are relational.

2.1.1.3 Integration Model: GAV and LAV

One of the main differences between a data integration system and a traditional database system is that users pose queries in terms of a mediated schema, not in terms of the database schema. The data is stored in the data sources, organized under local schemata. When the data integration system wants to answer queries posed on the mediated schema, there must be some descriptions of the relationship between the data sources and the mediated schema. These descriptions are known as schema mapping. The query processor of the integration system must be able to reformulate a query posed on the mediated schema into a query against the source schemata.

2.1.1.3.1 GAV: Global as View

Global as View (GAV) was introduced in [ACPS96], [FRV96], [CGMH$^+$97], [PAGM96]. In the GAV approach, for reach relation R in the mediated schema, a query is written over the source relations specifying how to obtain R's tuples from the sources. The advantage of GAV is that the query reformulation is relatively straightforward. The relations in the mediated schema are defined in terms of the source relations. The only work of query reformulation in GAV is to unfold the definitions of the mediated schema relations. The disadvantage of GAV is that it is difficult to add new data sources to the data integration system. It is necessary to consider all the possible interactions of the new participating data source with each of the existing ones. Separation of a local data source is costly. This disadvantage limits the extendibility of the data integration system with respect to new data sources.

2.1.1.3.2 LAV: Local as View

Local as View (LAV) was introduced in [DG97], [IFF$^+$99], [KW96], [LRO96]. Compared with GAV approach, in the LAV approach, the descriptions of the sources are given in the opposite direction. The contents of a data source are described as a query over the mediated schema relation. Each data source is described in isolation. It is the responsibility of the data integration system to figure out at query time how the sources interact and how their data can be combined to answer the query. The benefits of LAV are:

1. It is easier to describe the data sources because it is not necessary to incorporate knowledge from other data sources, nor to know about relationships between data sources. It is easier to add new data sources participating in the integration. This is very important when there are a lot of participating data sources and the number of participating data sources changes often.
2. The administrator of the integration system can write precise descriptions of the data sources by writing constraints on the content of the data sources. Therefore, it is easier for the data integration system to enhance query performance, e.g., to choose a minimal number of data sources relevant to a query.

The disadvantage of LAV is that query reformulation becomes more complicated than in GAV, because it is not possible to simply unfold the definitions of the relations in the mediated schema.

2.1.2 Relational vs. XML-based DIS

2.1.2.1 XML and XQuery

For a quarter century relational database systems have dominated the database industry. In relational database systems, the tables are flat. The order of information is not important and the data structure is quite stable and unchanged over time. However, since the Web arose, it has led to requirements for storage of new kinds of information in which the order of information is important and data structure can vary over time and from one document to another. These evolving requirements have given rise to Extensible Markup Language (XML) [W3C06a] as a widely accepted data format and to XQuery [BCF$^+$07] as an emerging standard language for querying XML data sources [OCKM06].

Since the first relational database systems appeared in the early 1980s, the commercial database field has seen many evolutionary changes. Most large-scale commercial database systems have been based on the relational data model and Structure Query Language (SQL) [SQL92].However, the appearance of XML and XQuery leads to a significant new approach to database management.

XML was developed by an XML Working Group (originally known as the SGML Editorial Review Board) formed under the World Wide Web Consortium (W3C) in 1996. XML has many advantages. For example, it can be used as a more expressive markup language as HTML, as an object-serialization format for distributed object applications, or as a data exchange format [FTS00]. XML is rapidly becoming one of the most widely adopted technologies for information exchange and representation especially in the Internet [BCP01].

In order to understand why XML can be used as a new approach to storing and retrieving data in the Internet, it is necessary to introduce the concept of *metadata*. Metadata is defined as "data about data". Metadata is the information that describes the structure of stored data. In relational database systems, data has a regular and repeating structure that can be described independently of any data instance. Normally in relational database systems, metadata is stored separately from the data itself, typically in a set of tables called the system catalog. The relational query languages make use of metadata when they process queries.

XML documents are self-describing data in which metadata is separated into two types: markup and schema. Markup contains information about individual instances of stored data. For example, a piece of data might be identified as an address or as a postal code. Schema, on the other hand, contains global information about how documents are assembled by the component parts. A schema for a purchase order, for example, might specify that a purchase order consists of a date, a customer, a ship-to address, an optional bill-to address, and an array of one or more items that in turn contain lower-level data structures. A schema for a given type of document specifies the degree of flexibility that is allowed in constructing documents of that type, such as alternative content, optional content, and constraints on the number of occurrences of various parts.

The first languages widely used in retrieving information from XML documents were XPath [CSD99] and XSLT [W3C99b]. XPath was designed as a notation for navigating within an XML document, which is structured as a hierarchy of elements and attributes. XPath can isolate the elements and attributes that satisfy a given search criterion, but it cannot construct a new element. For this reason, it is not a complete query language. XSLT is more powerful than XPath, but was designed primarily for transforming one document into another. The expressive power of XSLT is sufficient for a query language, but its recursive pattern matching paradigm is difficult to optimize and is better adapted for document transformation than for queries.

Recognizing the limitation of XPath and XSLT, the World Wide Web Consortium (W3C) organized a workshop in 1998 to begin the consideration of a new query language for XML data sources. One outcome of the workshop was the formation of a new W3C working group on XML Query, which has produced a draft specification of a new language called XQuery.

W3C XQuery [BCF+07] is a query language for XML. The most common use cases for XQuery involve XML publishing to create XML for Web messages, dynamic web sites, or publishing applications. The original data may be found in XML files, or it may be in a data store such as a relational database [Dat09].

2.1.2.2 Relational Data Integration Systems

Figure 2.3 shows a typical data integration system where the global data model is relational.

The data users are permitted to submit SQL queries and the query processor of the relational data integration system will parse the queries, rewrite the queries and divide the queries into sub-queries. The sub-queries are then transmitted to the wrappers. The wrappers are responsible for translating the sub-queries into sub-queries in the local query languages. The sub-queries in

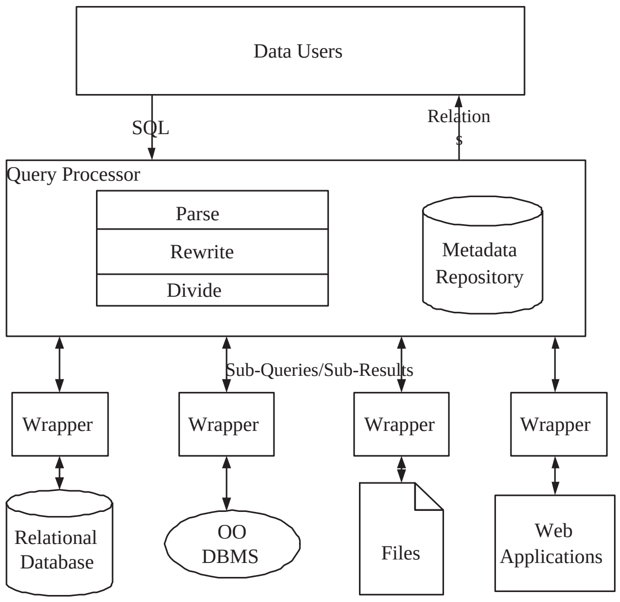


Fig. 2.3 A Typical Relational Data Integration System

the local query languages are then executed in each local data source. The results are then transmitted to the wrappers and the wrappers are responsible to translate the results into the results in the global data model. The query processor then combines all the sub-results and returns the results (normally relations, or tables) to the data users.

2.1.2.3 XML-based Data Integration Systems

XML-based data integration system has been studied and developed for many years and has become a mature technology [Car06]. Figure 2.4 shows a typically XML-based data integration system.

In an XML-based data integration system, the users or the end applications submit XML Queries to the data integration system. The query processor will parse the queries, rewrite the queries and divide them into sub queries. These sub queries are then transmitted to the wrappers of the XML-based data integration system. The wrappers will translate the sub queries given in XQuery syntax into the queries in the local query language and the sub queries in the local query language will be executed in the local data sources. The results are returned to the wrappers and the wrappers will translate the

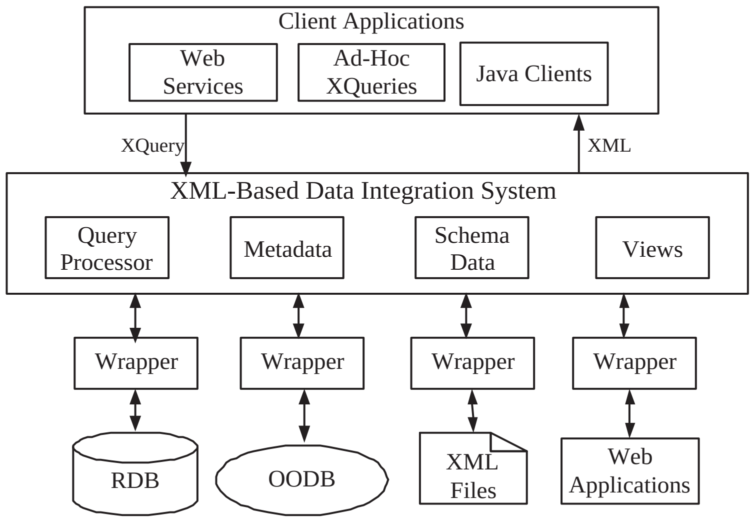


Fig. 2.4 A Typical XML-based Data Integration System

local results into XML format. The query processor then combines the sub results and returns them to the users or to applications.

2.1.2.4 Comparison of the Two Models

In order to state the necessity of an XML-based data integration system, we need to first compare it with the relational data integration systems. We first compare the two data models: XML data model and relational data model. Then we compare the two query languages used in the two data integration systems: SQL and XQuery.

2.1.2.4.1 Comparison of XML and Relational Model

First we list the difference of the two data formats.

1. Relational data is structured while XML data is semi-structured.
2. XML data is self-described while relational data is not.
3. XML data is often sparse while relational data is not.
4. Relational data has no intrinsic order that is independent of its values, but XML is often used to store intrinsically ordered data, such as paragraphs in a book.

5. Compared to a typical relational system catalog, XML schema [W3C06b] information is often more complex and supports structurally different XML data objects under the same schema. An XML query may operate over multiple documents conforming to different schemata or to multiple versions of a schema.
6. Relational databases represent information only by values, whereas XML also uses the concept of nesting (element hierarchies) and references.
7. XML mixes markup with data. Data and metadata are separated in relational databases.
8. XML documents often contain text, which increases the importance of search. Text search requires linguistic operations, such as stem matching, and often needs to combine precise with imprecise forms of search in a single query. Relevance ranking is an important form of search in XML data.

2.1.2.4.2 SQL and XQuery: Similarities and Differences

SQL is a mature relational database language that takes advantage of the regular structure of data stored in tables. [OCKM06] gives a detailed comparison between SQL and XQuery. The similarities between SQL and XQuery are:

1. Both are declarative query languages.
2. Both are functional languages defined in terms of a set of expressions that are closed under a specific data model.
3. The two languages are roughly equivalent in expressive power (both support joins, quantification, recursion, and user-defined functions).
4. Both languages have type systems that include simple and complex data types and inheritance.
5. Both languages have set-oriented operators, including union and intersection, as well as set-oriented search operators.

The differences between SQL and XQuery are:

1. XML data, unlike relational data, has an intrinsic order. So the design of XQuery must include positional predicates, "before" and "after" pred-

icates, and operators, such as path expressions, that preserve document order.

2. Because XML mixes markup with data, it is possible in XQuery to express queries that span both data and metadata. Data and metadata are separated in relational databases; therefore, SQL cannot express this kind of query.
3. XQuery has a concept of identity that is not present in SQL. In XQuery, nodes (which correspond to XML concepts such as elements and attributes) have identity, but atomic values (such as 47 and "Hello") do not. The concept of identity affects the XQuery language in several ways. The language has expressions called constructors that create new nodes with new identities. Also, XQuery set operators, such as union and intersect, and path expressions eliminate duplicates from their results based on node identity rather than on value, as in SQL.

2.2 Constraints and Triggers

The idea of integrity constraints in relational databases appeared not long after the relational model itself. From about the mid-1980's to the mid-1990's there were numerous research proposals and prototypes in the area of database constraints and triggers. In 1983, [CD83] developed an integrity monitor for the database system INGRES. The integrity subsystem guards the database against semantic errors by permitting users to make assertions which define the correctness of the database, and to specify actions to be taken when the assertions are not satisfied [EC75]. There are several kinds of assertions: tuple- vs. set-oriented assertions, state- vs. transition-oriented assertions, and immediate- vs. delayed assertions. Tuple-oriented assertions are normally about the individual tuples while set-oriented assertions may be based on built-in functions applied to sets, such as average, sum, maximum, etc, or may be based on the comparison of one set to another. The difference between state-oriented assertions and transition-oriented assertions is that to aid in state transition assertions, there must be the keywords such as "OLD" and "NEW". Immediate assertions mean that the assertions cannot be suspended but must be enforced even on intermediate stages of a transaction. Those assertions that is not declared to be “immediate” are delayed.

From then on most mainstream database products announced their support for constraints and triggers, with expressive constraint specifications in the SQL-92 standard [SQL92], and both constraints and triggers in the SQL-99 standard [Mel03].

Shortly after researchers recognized the importance of database integrity constraints, including automatic reactions to constraint violations, the idea expanded to the more general concept of triggers, also now known as Event-Condition-Action (ECA) rules or active rules [PD99].

2.2.1 Relational Constraints and Triggers

2.2.1.1 Relational Constraints in SQL

The SQL-92 (and subsequent SQL-99) standard provides several mechanisms for specifying integrity constraints. The most common kinds of constraints are keys, non-null constraints, and referential integrity. Among them referential integrity constraints can be specified with particular actions to be taken upon violations, such as cascaded delete or set null.

Check constraints are generally associated with a given database table. SQL-like syntax is used to specify conditions that must hold for each tuple of the table, and the conditions are checked on inserts and updates to the table. SQL-92 standard permits sub-queries within check constraints, thereby enabling check constraints to be used for multi-tuple and multi-table constraints. Unfortunately, most products do not support this feature. Those constraints which are not specific to a single table can be specified by SQL-92 assertions. Again many products do not support this level of generality. Finally, domain constraints can be specified to constrain all values in all columns of the domain, or any value cast to the domain.

When a constraint is first defined or when new data is loaded, the system verifies the constraint against the data. Thereafter the events are monitored and actions are executed to ensure that the database state always satisfies the constraint as specified.

There are four Constraint variations: **UNIQUE** constraints, **PRIMARY KEY** constraints, **FOREIGN KEY** constraints and **CHECK** constraints, which are defined in the following [Mel03]:

1. A **UNIQUE** constraint defines one or more columns of a table as unique columns; it is satisfied if no two rows in the table have the same nun-null values in the unique columns.
2. A **PRIMARY KEY** constraint is a **UNIQUE** constraint that specifies **PRIMARY KEY**; it is satisfied if

   - no two rows in the table have the same non-null values in the unique columns and
   - none of the primary key columns are NULL.

   UNIQUE constraints and **PRIMARY KEY** constraints describe a base table's candidate keys.
3. A **FOREIGN KEY** constraint defines one or more columns of a table as referencing columns whose values must match the values of some corresponding referenced columns in a referenced base table (the referenced columns must be **UNIQUE** columns for the referenced table). It is satisfied if, for every row in the referencing table, the values of the referencing columns are equal to those of the corresponding referenced columns in some row of the referenced table. If either table contains NULLs, satisfaction of the **FOREIGN KEY** constraint depends on the constraint's match type. **FOREIGN KEY** constraints describe linkages between base tables.
4. A **CHECK** constraint defines a search condition; it is violated if the result of the condition is **FALSE** for any row of the table. An **ASSERTION** is a **CHECK** constraint that may operate on multiple tables.

2.2.1.2 Relational Triggers in SQL

An SQL Trigger specifies a set of SQL statements that are to be executed, either once for each row or once for the whole triggering INSERT, DELETE, or UPDATE statement, either before or after rows are inserted into a table, rows are deleted from a table, or one or more columns are updated in rows of a table. Triggers are similar to constraints. In fact, referential constraints are now defined by the SQL standard as merely a type of trigger. What distinguishing a trigger from a constraint is the flexibility. The trigger body may contain actual SQL procedure statements that can be defined by the user.

Constraints are normally declarative while triggers are explicitly procedural. A trigger is activated whenever a specified event occurs. The event is usually an INSERT, DELETE, or UPDATE on a particular table. Once the trigger is activated, an optional specified condition is checked. If the condition is true, an action is executed. If the condition is omitted, it is considered as true and an action is executed, too.

Triggers have an activation time: BEFORE or AFTER. Triggers have a granularity: ROW-LEVEL or STATEMENT-LEVEL. There are transition variables which record the OLD and NEW value of the variables affected by the events. There are also transition tables: OLD_TABLE and NEW_TABLE. Each trigger has access to transition variables and transition tables. Conditions can be arbitrary predicates. Actions are stored procedures including SQL statements, control constructs, and calls to user-defined functions. The user-defined functions invoked by a trigger action may have side effects that fall outside of DBMS control.

There are two classic instances where triggers are used to support kernel database functionality: referential integrity and materialized views. In general, triggers can be classified into following categories [CCW00]:

1. Constraint-preserving triggers: Signal integrity constraint violations and force rollbacks of the violating transactions.
2. Constraint-restoring triggers: Detect integrity constraint violations and modify the database contents in order to restore integrity.
3. Invalidating triggers: Signal and mark integrity constraint violations, allowing applications to respond appropriately.

4. Materializing triggers: Compute materialized derived information, from simple scalar values to aggregate values to complex views, either by incremental modifications or complete refresh.
5. Metadata triggers: Maintain consistency across system catalogs or other metadata.
6. Replication triggers: Replicate, migrate, or log information and/or modifications from the primary copy of one table or database to the secondary copy.
7. Extenders: Manage new types of data and keep specialized external structures consistent with the base data.
8. Alerters: Notify or push information to users in the form of messages, typically based on a publish/subscribe model.
9. Ad-hoc triggers: Implement business rules, scheduling, workflow, supply-chain management, or other application-specific logic.

[CW91] [CW92] [CW93] [CFPT94] are the foundations of early work that triggers can be generated automatically for a wide class of applications: constraint maintenance, materialized view maintenance, and managing semantic heterogeneity.

2.2.2 XML Constraints and Triggers

2.2.2.1 XML Constraints

[LC00] presents the types of semantic constraints hidden in Document Type Definition (DTD) [W3C99a]. As DTD is now being replaced by XML Schema, we discuss XML constraints only in XML schema here. In general, there are altogether five kinds of XML constraints hidden in XML schema. In the following, we classify these constraints and give each kind of constraints an explaining example.

2.2.2.1.1 Domain Constraints

When the domain of the attributes is restricted to a certain specified set of values in XML Schema, it is called *Domain Constraints.* In Example 2.2.1, the domain of the Type ”myInteger” is restricted:

```
<xs:simpleType name="myInteger">
    <xs:restriction base="xs:integer">
      <xs:minInclusive value="-2"/>
      <xs:maxExclusive value="5"/>
    </xs:restriction>
  </xs:simpleType>
```

Example 2.2.1: Domain Constraints in XML Schema

2.2.2.1.2 Occurrence Constraints

Occurrence constraints define how many times the sub-element should appear. There are altogether four kinds of occurrence constraints. We use Example 2.2.2 to explain each of them:

1. 1-to-{0,1} mapping (“at most” semantics): An element may have either zero or one sub-element. (e.g., sub-element **ShipTo**).
2. 1-to-{1} mapping (“only” semantics): An element must have one and only one sub-element. (e.g., sub-element **BillTo**).
3. 1-to-{0, ...} mapping (“any” semantics): An element may have zero or more sub-elements. (e.g., sub-element **items**).
4. 1-to-{1, ...} mapping (“at least” semantics): An element may have one or more sub-elements. (e.g., sub-element **buyer**).

2.2.2.1.3 UNIQUENESS

Uniqueness constraints ensure that no duplicate values are entered in the specified elements. Example 2.2.3 defines a Uniqueness constraint. “@partnum” and “productName” must be unique in “items/item”.

```
<xsd:complexType name="PurchaseOrderType">
  <xsd:all>
    <xsd:element name="buyer"  type="Buyer"
         minOccurs="1" maxOccurs=unbounded/>
    <xsd:element name="shipTo" type="USAddress"
         minOccurs="0" maxOccurs="1"/>
    <xsd:element name="billTo" type="USAddress"
        minOccurs="1" maxOccurs="1"/>
    <xsd:element ref="comment" minOccurs="0"/>
    <xsd:element name="items"  type="Items"
        minOccurs="0" maxOccurs=unbounded/>
  </xsd:all>
  <xsd:attribute name="orderDate" type="xsd:date"/>
</xsd:complexType>
```

Example 2.2.2: Occurrence Constraints in XML Schema

```
<xsd:element name="items" type="Items">
  <xsd:unique name="partNumAndName">
    <xsd:selector xpath="item"/>
    <xsd:field    xpath="@partNum"/>
    <xsd:field    xpath="productName"/>
  </xsd:unique>
</xsd:element>
```

Example 2.2.3: UNIQUENESS Constraints in XML Schema

2.2.2.1.4 Keys

In relational database systems, a primary key is a column (or a set of columns, if you use the multiple-column constraint format) that contains a non-NULL, unique value for each row in a table. In XML Schema, “key” element is used to define, that the elements “a” under “root” element must have unique value of “id” attribute and it must be present (Not NULL). In Example 2.2.4, states are keyed by their codes.

```
<xs:key name="state">
  <xs:selector xpath=".//state"/>
  <xs:field xpath="code"/>
 </xs:key>
```

Example 2.2.4: KEY Constraints in XML Schema

2.2.2.1.5 Referential Constraints

In relational database systems, a referential constraint definition defines an integrity condition that must be satisfied by all the rows in two tables. The resulting dependency between two tables is affected by changes to the rows. In XML Schema, the referential constraint definition defines an integrity condition that must be satisfied by all the elements in the XML document. The resulting dependency between two elements is affected by the changes to the elements. Example 2.2.5 shows referential constraints in XML Schema.

```
<element name="purchaseReport">
    <complexType>
      <sequence>
        <element name="regions" type="r:RegionsType"/>

        <element name="parts" type="r:PartsType"/>
      </sequence>
      <attribute name="period"       type="duration"/>
      <attribute name="periodEnding" type="date"/>
    </complexType>

    <unique name="dummy1">
      <selector xpath="r:regions/r:zip"/>
      <field xpath="@code"/>
    </unique>

    <key name="pNumKey">
      <selector xpath="r:parts/r:part"/>
      <field xpath="@number"/>
    </key>

    <keyref name="dummy2" refer="r:pNumKey">
      <selector xpath="r:regions/r:zip/r:part"/>
      <field xpath="@number"/>
    </keyref>
  </element>
```

Example 2.2.5: Referential Constraints in XML Schema

To ensure that the part-quantity elements have corresponding part descriptions, we say that the number attribute (field xpath=”@number”/) of those elements (selector xpath=”r:regions/r:zip/r:part”/) must reference the “pNumKey key”. This declaration of number as a keyref does not mean that its value must be unique, but it does mean there must exist a pNumKey with the same value.

2.2.2.2 XML Triggers

XML is accepted as one of the most widely adopted standards for information exchange and representation. XML repository can be used both as direct data sources and as an interoperability layer between data sources.

Acting as interoperability layer means that data sources export XML views over their data, these XML views may then be materialized in the XML repository. Other systems can directly import or query these views stored in the XML repository. Many open source and commercial native XML databases have been developed, among which are Berkeley DB XML [Ora09], eXist [eXi08], MarkLogic [Mar08a], Tamino XML Server from Software AG [AG09], and Oracle XML DB [Ora08b]. Like relational database systems, these XML database systems support not only data storage but also query processing mechanisms and indexing mechanisms. Several XML query languages have been developed to support querying in XML database systems: Lorel [AQM$^+$97], XML-QL [DFF$^+$98], XQL [RLS$^+$98], and XQuery [BCF$^+$07]. XQuery is the standard query language proposed by W3C. In order to evolve XML into a universal data representation and sharing format, several XML update languages have been proposed [TIHW01] [XML00]. The XQuery update operations commonly include insertion, deletion, replacement, and renaming of XML data.

After extending XML query language with update capabilities, it is necessary to provide a mechanism to guarantee the data integrity in XML repository. In order to make XML repository fully equipped with data management capabilities, XML repository must also be extended to support active rules for checking the correctness of updates and monitoring changes of documents.

As in SQL triggers, an XML trigger also mainly includes three parts: events, conditions and actions. We will explain each of these three parts in the following.

2.2.2.2.1 Event

The event part of the XML trigger specifies the events responsible for rule triggering. A mutating event is generated when the XML content is modified.

We assume three types of *mutating* events: *insert*, *delete*, and *update*. The definition of a mutating event declaratively describes the nodes (elements, or attributes) whose modifications need to be monitored. Every time a monitored modification occurs, the corresponding event instance is generated and associated with the modified node, e.g., an event definition *insert*(*//house*) monitors the insertion of "house" elements in the repository; an event instance is generated whenever a "house" element is introduced.

The event specifies what causes the rules to be triggered. Useful triggering events are:

- Data modification. The event is specified as a change to an XML document, resulting in an invocation of a data manipulation primitive within an XML repository. Primitives for data modification include *insert*, *delete* and *update* of XML information items, either performed manually by the user or accomplished through an update language. In addition, an event might be specified as the invocation of a function or a particular method that modifies the XML data.
- Data retrieval. The event might be specified as a query that extracts XML items from the repository or as an invocation of a method or function that retrieves XML data.
- Application-defined. The event might be specified as high-level operation of Web or document management applications, such as the publishing or updating of a Web page or the opening or closing of an editing session upon a document, the sending of a document through a mailer or the user connection to a Web site. These events are generated by applications external to the XML repositories and can be possibly detected without accessing it. A name is declared to denote such events (defined as constants, such as user-login, publish-doc, open-session, send-doc, etc.)

2.2.2.2.2 Condition

The *condition* part of the XML triggers specifies the predicate that must be satisfied to execute the trigger's action, expressed through a query which is interpreted as a truth value if it returns a nonempty answer. An important feature is the presence of a communication mechanism between the condition and the event part, so that the condition has a way to refer to the nodes on which the events occurred. This communication is based on predefined variables \$*old* and \$*new* that represent the nodes on which the events occurred with their current and past values, in a way similar to transition variables of database triggers. Possible conditions include:

- Predicates. The condition might specify that a certain predicate holds on the XML repository. This predicate is encoded by using the condition clauses of an XML Query language. Conditions can be arbitrarily complicated as they might include full-fledged joins and aggregates. When the condition evaluation is very expensive, conditions may be restricted to support only comparison operators (restricted predicates).
- Queries. The condition might be a full query onto the XML repository. The condition is evaluated to be true whenever the query produces a non-empty answer. A generic query inspects the content of the repository in its current or past states.
- Methods. A rule condition might be specified as a call to a function written in an external programming language or in the query language itself, if supported. If the method invocation returns true, then the condition holds. Moreover, the method can be a SOAP invocation, shipped to a remote XML server.

2.2.2.2.3 Action

The *Action* part of a rule specifies a method to invoke when the rule condition is evaluated true. Possible actions include:

- Data modification operations. Insert, delete and update operations are allowed as rule actions.
- Data retrieval. The action might be a query on the XML repository to retrieve data, or a function or method devoted to retrieve data.
- Application methods. A rule action might be the invocation of a method, either expressed in a programming language or in the query language to access the repository, or encoded in SOAP and shipped on the network.

2.2.2.2.4 Transition Values

The triggering of a rule occurs within a database state change, that is called *transition*. A transition can be limited to a small change on the database, or can be expanded to cover an entire transaction. We suppose that these concepts are available in an XML repository, and consider the evaluation of rules to be held within transactions as well. Transition values are designated as the references to *old* and *new* data after modifications to the repository.

2.3 Semantic Query Optimization

In the early 1980's, researchers recognized that semantic information stored in databases could be used for query optimization, too [SO89] [CGM90] [JCV84]. A new set of techniques called semantic query optimization (SQO) was developed. SQO adds a new and much wider dimension to the process of conventional query optimization. Instead of just using syntactic knowledge, it also uses the integrity constraints associated with a database to improve the efficiency of query evaluation, such as knowledge about the domains of relations, nature of data, etc. Integrity constraints (ICs) are important to express the semantics in databases for a long period of time. They express the properties that must be true about the data stored in a database. Once declared, the database system must protect the integrity of the database as expressed by the integrity constraint, if it is activated.

With semantic query optimization more efficiency can be achieved for query optimization than by only using syntactic query optimization techniques. This is because syntactic optimizer does not understand semantic knowledge, which should be satisfied by all instances of particular database In this case syntactic optimizer often leads to a suboptimal form for the given query for execution. Certain queries, which can be answered without any relation scan in the database, cannot be detected by a syntactic optimizer, thus resulting in superfluous database accesses. Syntactic optimizers cannot detect and eliminate semantically redundant restrictions or joins in the queries or introduce some useful restrictions and joins into the query to reduce the overall cost of query execution, either.

Two queries are semantically equivalent if they return the same answer for any database state satisfying a given set of integrity constraints. A semantic transformation transforms a given query into a semantically equivalent one. Semantic query optimization is the process of determining the set of semantic transformations that result in a semantically equivalent query with reduced execution costs.

In [CGK+99], the authors outline five common SQO techniques:

- Join elimination: A query contains a join for which the result is known as a priori, hence it does not need to be evaluated. For example, when the two attributes of the join are related by a referential integrity constraint. (A.a= B.a, A.a is referenced by B.a.)
- Join introduction. It may be advantageous to add a join with an additional relation, if that relation is relatively small compared to the original relations as well as highly selective. If the join attributes are indexed, higher benefit will be generated. This is under the assumption that the retrieval of tuples using an index is more efficient than their sequential processing. But in general, this is not always true.
- Predicate elimination: if a predicate is known to be always true it can be eliminated from the query.
- Predicate introduction: a new predicate on an indexed attribute may result in a more efficient access methods. Similarly, a new predicate on a join attribute may reduce the cost of the join as well as the size of the join result.
- Detecting the empty result: if the query predicate is inconsistent with the integrity constraints, the query does not have an answer. Similarly, if a predicate evaluates to false, the query's result is set empty.

Unfortunately, although semantic query optimization is well known to be useful and could potentially provide much greater performance improvements than traditional syntactic query optimization, only few commercial products employed it. In [CGK+99], the authors present three reasons why SQO has never caught up in the commercial world where most databases are RDBMS:

- SQO is in many cases designed for deductive databases where the relatively high cost of applying complex rules (in comparison to much less complex rules in relational databases) is more likely to make the extra computational effort for SQO worthwhile. Because of this association, SQO might not appear useful for relational database technology.
- At the time when SQO techniques were developed, the CPU and I/O speeds were not high enough for the extra computational cost of SQO to be acceptable. The savings in query execution time (dominated by I/O) that SQO could provide was not worth the extra CPU time necessary to optimize a query semantically.
- ICs usually have to be defined at first for a given database, and then SQO may be applied. Although many ICs in fact exist in the most real life databases, however, only few of them are ever defined.

Nowadays the computer hardware has dramatically involved. The speed of CPU and I/O has been greatly improved. The extra processing for SQO seems to be acceptable now. At the same time the computer storage capability has been improved and currently there is a great volume of data existing in current enterprise information systems, which makes data retrieval more time-consuming. After distributed database and data integration systems appeared, network delays in answering queries appeared as well. Under this situation, using integrity constraints to semantically optimize queries to the data integration system becomes more attractive. This work gives an approach to semantically optimize queries to an XML-based data integration system exploiting the constraints coming from the local data sources.

2.4 Active Database System

Active database systems support mechanisms that enable them to respond automatically to events that are taking place either inside or outside the database system itself [PD99]. Active databases support their applications by moving the reactive behavior from the application (or polling mechanism) into the DBMS. Active databases are thus able to monitor and react to specific circumstances of relevance to an application. The reactive semantics are both centralized and handled in a timely manner. An active database system must provide a knowledge model (i.e., a description mechanism) and an execution model (i.e., a runtime strategy) for supporting this reactive behavior. A common approach for the knowledge model uses rules that have up to three components: an event, a condition, and an action. The event part of a rule describes a happening to which the rule may be able to respond. The condition part of the rule examines the context in which the event has taken place. The action describes the task to be carried out by the rule if the corresponding event has taken place and the condition has evaluated to be true.

Most active database systems support rules with all three components described; such a rule is known as an event-condition-action or ECA-rule. In some proposals the event or the condition may be either missing or implicit. If no event is given, then the resulting rule is a condition-action rule, or production rule. If no condition is given, then the resulting rule is an event-action rule.

At first glance, the introduction of active rules to a database system may seem like a straightforward task, but in practice proposals must be made because widely different functionalities must be supported. Among the issues that distinguish the proposals are the expressiveness of the event language, the scope of access to database states from the condition and action, and the timing of condition and action evaluation relative to the event. The functionality of a specific system will be influenced by a number of factors, including the nature of the passive data model that is being extended, and the categories of application to be supported.

When ECA rules are put into a distributed environment, things become more difficult. How to detect distributed events, how to evaluate conditions in a distributed environment, how to coordinate the action part with the event part and how to coordinate the different parts in the action part in a distributed environment are problems that are not yet fully solved and need more research effort. We go a step further in this direction and try to settle this problem in a data integration system.

2.5 Summary

Data integration has long been recognized as the kernel problem in database research and industry. At the beginning, data integration systems were relational. That means, the data model in the integration level is relational. As XML has become the main technological ingriedents in Internet data exchange, XML-based data integration systems appeared. In this chapter, we first discussed the fundamentals of data integration. Then we compared the two integration models: relational model and XML model. The different query languages in these two kinds of data integration system, SQL and XQuery were compared according to similarities and differences. After that we introduced the foundations of constraints and triggers. These two concepts appeared soon after the database came into the world. We first introduced the relational constraints and triggers especially in SQL. Then the XML constraints and triggers were discussed. This knowledge forms the fundamental of this work. Our goal is to use constraints and triggers to enhance an XML-based data integration system in three ways. First, to use constraints to semantically optimize queries submitted to XML-based data integration systems. Second, to use the constraints to ensure data integrity in XML-based data integration systems. Third, to use the triggers to enforce data consistency among the data sources of the XML-based data integration systems. So, fundamentals of semantic query optimization and active database system were discussed at the end of this chapter.

Chapter 3

Query Optimization by Constraints

The first enhancement by constraints in our work is to semantically optimize the queries submitted to the data integration system. In this chapter, we mainly introduce how to use constraints to improve the query performance of the data integration system. We will begin by a motivation example. Then we will analyze the difficulties to express different constraints coming from heterogeneous data sources. Our approach will follow including a uniform constraint model, constraint wrapper, and the semantic query optimization techniques. We will present the performance of the semantic query optimizer and give some discussion at the end.

3.1 Motivating Example

In order to give a more plain and clarified explanation of the ideas in this chapter, we first assume an XML-based data integration system as shown in Figure 3.1.

Suppose a construction company keeps data about a building under construction in XML files. This data must be consistent with the architect's design, which is in an RDBMS stored in the architect's computer.

The door information is stored in Table 3.1 by the architect.

Table 3.1 Door Information in the Architect Database

DoorNr	Width	Length
001	1.5	3
002	1.8	2.8

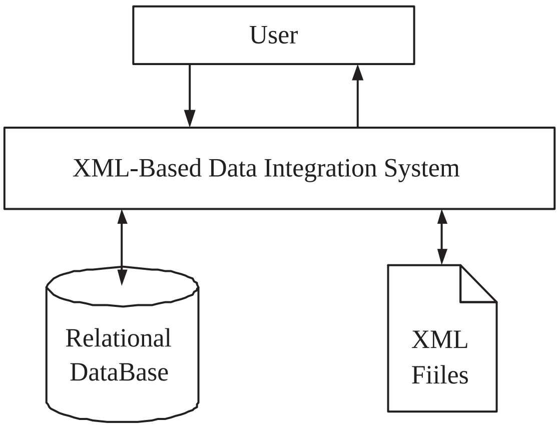


Fig. 3.1 An Assumed XML-Based Data Integration System

The information of the doors is stored in the following XML file by the construction company:

```
<door_info >
      <door>
        <door_nr>001</door_nr>
        <width>1.5</width>
        <length>3</length>
      </door>
      <door>
        <door_nr>002</door_nr>
        <width>1.8</width>
        <length>2.8</length>
      </door>
</door_info>
```

Example 3.1.1: DoorInformation.xml

Suppose there is a constraint in this data integration system as the following:

Constraint 1: The length of the door is less than 3.2 meters, which is a relational constraint:

```
CONSTRAINT P CHECK (length < 3.2)
```

When the user submits the following query:

```
Find a door whose length is more than 3.3 meters.
```

This query will be pushed down to the relational wrapper and translated into SQL as follows:

```
SELECT door_nr FROM door_info WHERE length>3.3
```

There will be an empty result set returned. If **Constraint 1** would be stored at the integration level, the data integration system would be able to detect that the length of the doors in the relational table will be less than 3.2. This query will return an empty result set. It should not be pushed down to the relational data source nor be translated into SQL. There is no need to open and scan the relational table.

When the user submits the following query:

```
Find a door whose length is less than 3.3 meters.
```

This query will be pushed down to the relational wrapper and translated into SQL as follows:

```
SELECT door_nr FROM door_info WHERE length<3.3
```

If the **Constraint 1** would be stored in the integration level, the data integration system would be able to detect that the length of the doors in the relational table will be less than 3.2. So the predicate $''length \leq 3.3''$ always holds. This predicate can be eliminated. There is no need to compare the value of the length attribute with 3.3. This is especially significant when the local data source lacks this optimization mechanism locally.

3.2 XML Constraints

If we want to use the local constraints from the data sources to semantically optimize the queries submitted to the data integration system, the first problem is how to express these local constraints in the data integration layer. The first step is to consider whether XML Schema can express all these local constraints.

3.2.1 XML Schema and Constraints

Constraints for semi-structured data have been the topic of research for several years. [BFSW01] introduces various kinds of semi-structured data constraints, including relative key constraints, path inclusion constraints, and set-valued foreign keys. [Bon02] also discusses the interactions among them.

However, current schema languages support integrity constraints only to a limited extent. XML Schema is capable of expressing fixed-format structural and type constraints, but it does not support generic constraints. XML Schema supports structured constraints (for the assessment of document well-formed-ness w.r.t. to the XML information items) as well as type and identity constraints. Thus, in an XML-based data integration system, the global schema, which is normally XML Schema, can express some of the constraints found in typical heterogeneous data sources, e.g., Primary Key, Foreign Key and Unique from Relational Database; OID, ObjectKey, Referential Integrity from Object-Oriented Database; etc.

3.2.2 Three Approaches to Express Constraints

Unfortunately, there are many other constraints, which cannot be expressed by XML Schema. For example: CHECK, ASSERTION and TRIGGER from relational databases; semantic constraints such as $C1 > C2$; etc. [xfr06] presents three different approaches to express those constraints that cannot be expressed by XML Schema directly:

- Supplement with another schema language (e.g., DSD [DSD08], DTD [W3C99a], Schematron [Sch09], Relax NG [MUR09], etc) which can support constraints in a comparable way.
- Writing code (e.g., Java, C++).
- Using XSLT /XPath Stylesheet.

We will analyze each of these approaches to see whether it fits for expressing local constraints at data integration layer.

3.2.2.1 Supplement with Another Schema Language

For a better understanding of the first approach we need the following definitions:

Definition 1. A grammar-based schema language specifies what elements may be used in an XML instance document, the order of the elements, the number of occurrences of each element, and the content/data type of each element. It specifies what components are allowed and the rules for using the components.

Relax NG [MUR09], XML Schema, and DTD are grammar-based schema languages. Relax NG and XML schema use XML syntax. DTD uses a non-XML syntax.

Definition 2. An assertion-based schema language specifies the relationships between the elements and attributes in an XML instance document. It is also called a rule-based schema language.

Schematron [Sch09] is an assertion-based language and it uses XPath to express assertions. Schematron provides meaningful, user-defined diagnostic messages. It is the only schema language with support for connecting a data check to a diagnostic.

Figure 3.2 shows the procedure to check whether an XML document is valid or invalid with the help of Schematron.

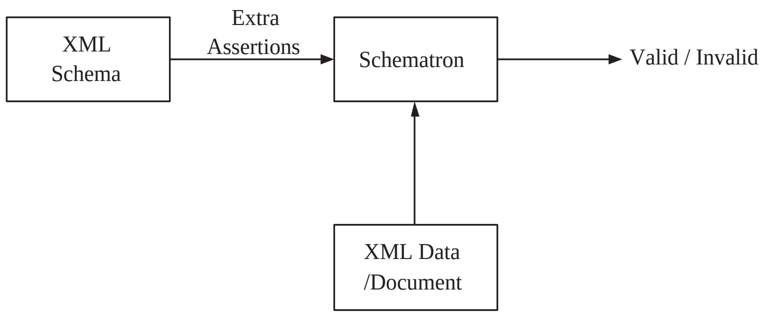


Fig. 3.2 Supplement with Schematron

Using Schematron the additional constraints (as "assertions") are embedded within the schema document (within "appinfo" elements), as shown in Example 3.2.1. This example shows how to use Schematron to express a constraint 'A is greater than B', which cannot be expressed by XML Schema.

A Schematron engine will then extract the assertions and validate the instance document against the assertions.

```
<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
        targetNamespace="http://www.demo.org"
        xmlns="http://www.demo.org"
        xmlns:sch="http://www.ascc.net/xml/Schematron"
            elementFormDefault="qualified">
<xsd:annotation>
<xsd:appinfo>
    <sch:title>Schematron validation</sch:title>
    <sch:ns prefix="d" uri="http://www.demo.org"/>
</xsd:appinfo>
</xsd:annotation>
<xsd:element name="Demo">
<xsd:annotation>
<xsd:appinfo>
        <sch:pattern name="Check A greater than B">
        <sch:rule context="d:Demo">
            <sch:assert test="d:A &gt; d:B"
             diagnostics="lessThan">
            A should be greater than B
            </sch:assert>
        </sch:rule>
        </sch:pattern>
  <sch:diagnostics>
  <sch:diagnostic id="lessThan">
     Error! A is less than B.
    A = <sch:value-of select="d:A"/>
     B = <sch:value-of select="d:B"/>
  </sch:diagnostic>
  </sch:diagnostics>
</xsd:appinfo>
</xsd:annotation>
<xsd:complexType>
<xsd:sequence>
        <xsd:element name="A" type="xsd:integer"
           minOccurs="1" maxOccurs="1"/>
        <xsd:element name="B" type="xsd:integer"
           minOccurs="1" maxOccurs="1"/>
</xsd:sequence>
</xsd:complexType>
</xsd:element>
</xsd:schema>
```

Example 3.2.1: An XML Schema with Schematon

Figure 3.3 shows the architecture for determining if your data meets all constraints: The XML data and the XML schema are put into the schema validator to see whether there is invalidity. If there is none, then the document is checked as valid by the schema validator. The XML schema and the XML data are then transformed to Schematron to check the assertions to see whether there is conflict or violation. If there is no violation, the XML data

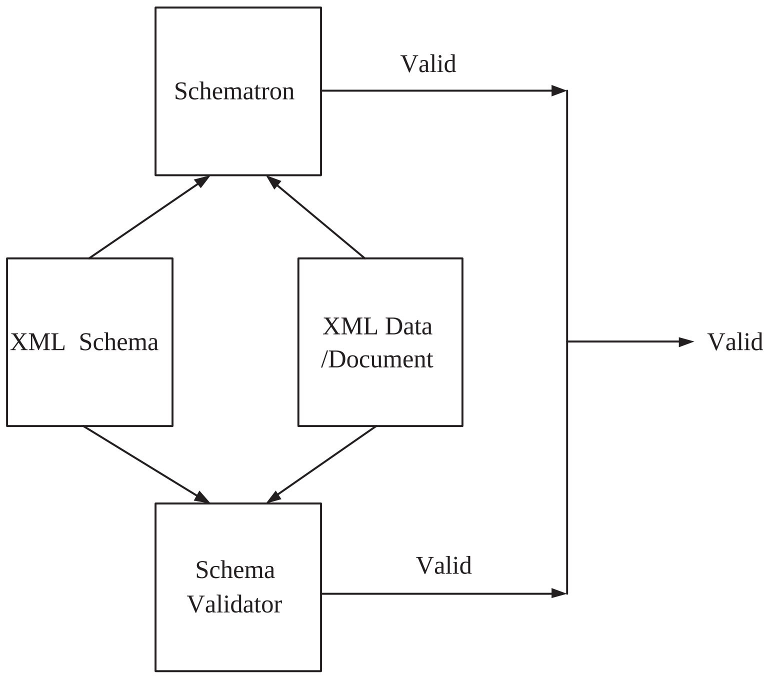


Fig. 3.3 Supplement with Schematron: Architecture

is considered to be valid. Only if both checks consider the data valid, then the data is valid.

The advantages of a supplement with another schema language are:

1. The constraints are collocated.
2. Many of the schema languages were created in reaction to the complexity and limitations of XML schemata. Consequently, most of them are relatively simple to learn and use.

The disadvantages of a supplement with another schema language are:

1. Multiple Schema Languages Required: Each schema language has its own capabilities and limitations. Multiple schema languages may be required to express all the additional constraints.
2. Yet Another Vocabulary: There are many schema languages, each with its own vocabulary and semantics. Although relatively easy to learn and use, it still takes time to learn a new vocabulary and semantics.
3. Questionable Long Term Support: In most cases the schema languages listed above were created by a single author. These products normally lack of long term support.

3.2.2.2 Writing Code

The second approach is to check the constraints by writing code through high-level programming languages, such as C++, Java, etc. The advantage of this option is that with a single programming language you can express all the additional constraints. The disadvantage is that one must go through the compiling, linking, executing effort. Another disadvantage of this option is that constraints are embedded in the code and are not in an abstract expression. It is hard to understand and maintain the constraints.

3.2.2.3 Using XSLT/XPath

Figure 3.4 shows the architecture for writing a stylesheet to check the constraints. Similar with supplement with Schematron, the XML data and XML Schema are put into the schema validator to check the validity. The XML data and the stylesheet, with the constraints given as code, are put into the XSL processor. The XSL processor will execute the code to see whether the XML data conforms to the constraints. Only when the result from the schema validator and the result from the XSL processor are both valid, the XML data is considered valid.

Example 3.2.2 shows the stylesheet to check the constraints.

The advantages of writing constraints as XSLT stylesheet are:

1. Application Specific Constraint Checking: Each application can create its own stylesheet to check constraints that are unique to the application.
2. Core Technology: XSLT/XPath are core technologies which are well supported, well understood, and with lots of material written on it.
3. Expressive Power: XSLT/XPath is a very powerful language. Most constraints can be expressed using XSLT/XPath. Thus you don't have to learn multiple schema languages to express your additional constraints.
4. Long Term Support: XSLT/XPath is well supported by W3C.

The disadvantages of writing constraints as XSLT stylesheet are:

1. Separate Documents: With this approach it is necessary to write the XML Schema document and a separate XSLT/XPath document to express ad-

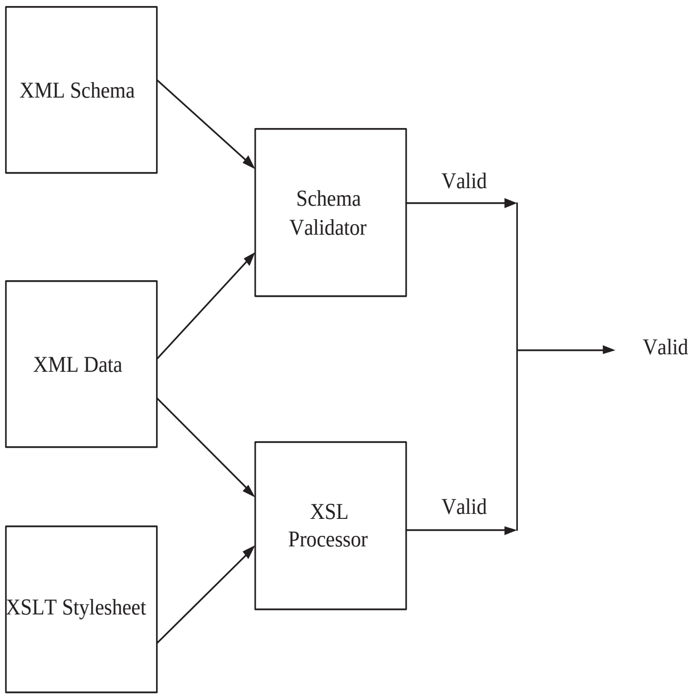


Fig. 3.4 Writing a Stylesheet to Check the Constraints. The XSLT Stylesheet Contains Code to Check Additional Constraints

```
<?xml version="1.0"?>
  <xsl:stylesheet
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    xmlns:d="http://www.demo.org"  version="1.0">
<xsl:output method="text"/>
<xsl:template match="/">
        <xsl:if test="/d:Demo/d:A &lt; /d:Demo/d:B">
        <xsl:text>Error! A is less than B</xsl:text>
                <xsl:text>&#xD;&#xA;</xsl:text>
                 <!-- carriage return -->
                <xsl:text>A = </xsl:text>
                        <xsl:value-of
                            select="/d:Demo/d:A"/>
                <xsl:text>&#xD;&#xA;</xsl:text>
                <!-- carriage return -->
                <xsl:text>B = </xsl:text>
                        <xsl:value-of
                            select="/d:Demo/d:B"/>
        </xsl:if>
        <xsl:if test="/d:Demo/d:A &gt;= /d:Demo/d:B">
                <xsl:text>
                  Instance document is valid
                </xsl:text>
        </xsl:if>
</xsl:template>
</xsl:stylesheet>
```

Example 3.2.2: Sample Constraint Expressed in XSLT ditional constraints. Keeping the two documents synchronous needs to be carefully managed.

2. XSLT is designed for document formatting rather than toward querying. The rule-based syntax of XSLT is quite awkward as a language for expressing constraints. Also, using XSLT one cannot express updating a document directly, because it requires a new result tree to be generated by applying transformation to the source document [BPW02].

3.3 A Uniform Constraint Model

Section 3.2 analyzed the possibilities to use XML Schema together with another schema language or programming code or XSLT stylesheet to express all the constraints. Each approach has its own advantages and disadvantages. When we put the whole scenario into the data integration system where only the virtual XML views exist in the integration level, it is not suitable to use any of the above three approaches. The ultimate goal of a data integration system is to make the user free from the tedious tasks of learning a new schema language, or writing code such as programming code or even XSLT code. But supplement with any other technologies will force the users and the developers to learn the new technologies. The second approach, writing the constraints by programming code, makes the maintenance of constraints very hard: everytime when there is an update of constraint, the developers have to go into the code and to modify the programs. After that they need to compile, debug, and run. The last one, writing the code as XSLT stylesheet, the constraints are also embedded in the programs. It is hard to know what constraints exist in the data integration system. In general, the third one is not an abstract-level constraint expressing mechanism.

We try to find a constraint model, which is abstract and simple enough for the developers to learn and which has won popularity and commercial use and that should have better properties than any of the above three.

We observe the fact that an important usage of triggers is to check integrity constraints. We go further and try to use triggers as the constraint model. This trigger constraint model integrates most of the advantages given in Section 3.2. But until now there is no standard for XQuery triggers available. Fortunately, in the research field there has been many work on XQuery triggers, among them Active XQuery has won great popularity and usage value. We first give a short introduction on Active XQuery in Section 3.3.1. Then we show how to use an abstract and simplified Active XQuery model to express the constraints in the data integration system. We also use the Active XQuery-based model to express the triggers in the data integration system to implement the business rules, which is the original usage value of the triggers, as will be shown in Chapter 5.

3.3.1 Active XQuery

Active XQuery was first proposed by [BBCC02] and was popularly used in many XML trigger scenarios, e.g. [SNS06]. Active XQuery was first designed as active rules (triggers) for XML repositories.

The syntax is shown in Example 3.3.1.

```
1:  CREATE TRIGGER Trigger-Name
2:  [WITH PRIORITY Signed-Integer-Number]
3:  [BEFORE| AFTER]
4:  (INSERT|DELETE|REPLACE|RENAME)+
5:     OF XPathExpression (,XPathExpression)*
6:  [FOR EACH (NODE|STATEMENT)
7:  [ XQuery-Let-Clause ]
8:  [ WHEN XQuery-Where-Clause]
9:  DO (XQuery-UpdateOp|ExternalOp)
```

Example 3.3.1: Active XQuery Syntax

The meaning of each line is explained in the following:

- Line 1: The CREATE TRIGGER clause is used to define a new XQuery trigger, with a specified name.
- Line 2: Rules can be prioritized in an absolute ordering, expressed with an optional WITH PRIORITY clause, which admits as argument any signed integer number. If this clause is omitted, the default priority is zero.
- Line 3: The BEFORE/AFTER clause expresses the triggering time relative to the operation.

- Line 4: Each trigger is associated with a set of update operations (insert, delete, rename, replace), adopted from the update extension of XQuery from [TIHW01].
- Line 5: The operation is relative to elements that match an XPath expression (specified after the OF keyword), i.e. a step-by-step path descending the hierarchy of documents. One or more predicates (XPath filters) are allowed in the steps to eliminate nodes that fail to satisfy given conditions. Once evaluated on document instances, the XPath expressions result into sequences of nodes, possibly belonging to different documents.
- Line 6: The optional clause FOR EACH NODE/STATEMENT expresses the trigger granularity. A statement-level trigger executes once for each set of nodes extracted by evaluating the XPath expressions mentioned above, while a node-level trigger executes once for each of those nodes. Based on the trigger granularity, it is possible to mention in the trigger the transition variables:

  1. If the trigger is node-level, variables OLD_NODE and NEW_NODE denote the affected XML element in its before and after state.
  2. If the trigger is statement-level, variables OLD_NODES and NEW_NODES denote the sequence of affected XML elements in their before and after state.

- Line 7: An optional XQuery-Let-Clause is used to define XQuery variables whose scope covers both the condition and the action of the trigger. This clause extends the "REFERENCING" clause of SQL, because it can be used to redefine transition variables.
- Line 8: The WHEN clause represents the trigger condition, and can be an arbitrarily complex XQuery Where clause. If omitted, a trigger condition that specifies WHEN TRUE is implicit.
- Line 9: The action is expressed by means of the DO clause, and it can be accomplished through the invocation of an arbitrarily complex update operation. In addition, a generic ExternalOp syntax indicates the possibility of extending the XQuery trigger language with support to external operations, permitting, e.g., to send mail or to invoke SOAP procedures.

3.3.2 Uniform Constraint Model

One of the most important usage of triggers is to enforce integrity constraints. We use triggers to develop the uniform constraint model for data integration systems to express all kinds of constraints from heterogeneous local data sources.

We choose to use Active XQuery [BBCC02] as the base of our constraint model for the following reasons: first, the syntax of Active XQuery is simple; second, it can be automatically invoked in response to events; third, its execution model is simple enough for rapid prototyping of a wide rage of usage. Fourth, it is capable of expressing both local constraints and global domain constraints and the global referential constraints. Here is the uniform constraint model based on Active XQuery in Figure 3.5.

```
1:    DECLARE NAMESPACE ns1 (,ns)*,
2:    CREATE CONSTRAINT Constraint-Name
3:    ON    (INSERT|DELETE|REPLACE)+
4:    OF XPathExpression (,XPathExpression)*
5:    [FOR EACH (NODE|STATEMENT]
6:    [XQuery-Let-For-Clause]
7:    [WHEN XQuery-Where-Clause]
8:    DO
9:         ERROR Message
           | Other Action
```

Fig. 3.5 Constraint Model

The first line is the namespace definition. Which data sources and which data objects in the data integration system are associated in the constraint definition are listed in the namespace declaration. The second line is the constraint name. The third line gives the constraint-associated operations. The fouth line is the constraint-relative elements. The fifth line is the constraint granularity. The sixth line defines the XQuery variables covering both the condition and the action. The seventh line is the constraint checking condition and the eighth line is to send the error message to the end application or to take some actions.

This uniform constraint model adheres to the spirit and practice of trigger definition and execution model of the SQL standard, which has gained tremendous popularity for developing data-intensive applications and which is the most used in commercial systems [BBCC02]. This constraint model is powerful enough to express all the constraints coming from heterogeneous data sources and the global constraints in the data integration system in the global level.

3.4 Constraint Wrapper and Constraint Repository

In order to make the data integration system constraint service understand, the constraints from typically heterogeneous data sources should be translated into constraints in the uniform constraint model at the global level in a global schema. Our constraint wrappers borrow the ideas from data integration wrappers. We have relational constraint wrapper for relational constraint mapping, XML wrapper for XML files, etc. We also provide a special way for those data sources which lack constraint mechanisms to submit constraints. These data sources include web services, HTML files, text files, etc. We permit them to submit predicate-like constraints. Constraint mapping depends on schema mapping. When translating constraints, the data integration system must provide schema mapping information. After the translation by the constraint wrapper, the constraints are expressed in the uniform constraint model and in the global schema. The translation flow can be simply expressed as depicted in Figure 3.6 [LM07].

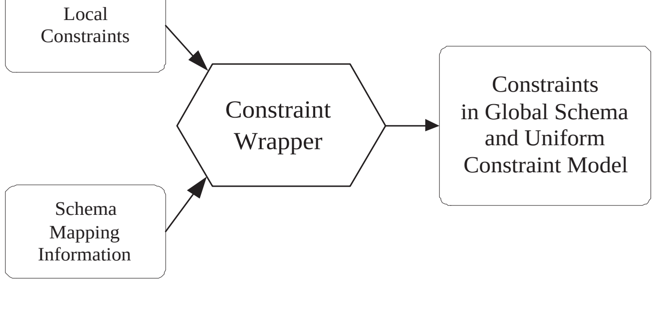

Local Constraints
Schema Mapping Information
Constraint Wrapper
Constraints in Global Schema and Uniform Constraint Model

Fig. 3.6 Architecture of Constraint Wrapper

In the next sections, we will give the examples to show how heterogeneous constraints and global constraints are expressed in the uniform constraint model.

3.4.1 Expression of Relational Constraints

3.4.1.1 CHECK

Example 3.4.1 is a CHECK definition.

```
    CREATE TABLE movie_titles(
       our_dvd_cost INTEGER
            CHECK   (our_dvd_cost < 90),
      );
```

Example 3.4.1: CHECK CONSTRAINT

The schema mapping information is shown in Table 3.2.

Table 3.2 Schema Mapping of Table movie_titles

Original Type	Original Name	Global Name
Table	movie_title	document('movie_titles.xml')
Column	our_dvd_cost	document('movie_titles.xml')/our_dvd_cost

This constraint will be expressed in the uniform constraint model as Example 3.4.2 and Example 3.4.3 shows.

```
        DECLARE NAMESPACE ns1=document('movie\_titles.xml')
        CREATE CONSTRAINT  check_our_dvd_cost
        ON INSERT OF ns1/our_dvd_cost
        Let $cost:= ns1/our_dvd_cost
        WHEN  $cost>= 90
        DO (PopupErrorMessage("our dvd cost must be less
                than 90"))
```

Example 3.4.2: CHECK Constraint in Uniform Constraint Model by INSERT Operation

All the "CHECK" constraints can be translated in this way.

```
        DECLARE NAMESPACE ns1=document('movie\_titles.xml')
        CREATE CONSTRAINT  check_our_dvd_cost_update
        ON UPDATE OF ns1/our_dvd_cost
        Let $cost:= ns1/our_dvd_cost
        WHEN  $cost>= 90
        DO (PopupErrorMessage("our dvd cost must be less
                than 90"))
```

Example 3.4.3: CHECK Constraint in Uniform Constraint Model by UPDATE Operation

3.4.1.2 ASSERTION

Example 3.4.4 is an ASSERTION definition across two tables guaranteeing that the payment of a DVD is received before the due date.

```
        CREATE ASSERTION receivedEqualsDue
        CHECK (not exist(SELECT * FROM dvd_sell ds,  dvd_order do
         WHERE  ds.payReceived>do.payDue AND
                     Ds. name=do. name) )
```

Example 3.4.4: ASSERTION

The schema mapping information is shown in Table 3.3:

Table 3.3 Schema Mapping of Table dvd_sell and dvd_order

Original Type	Original Name	Global Name
Table	dvd_sell	document('dvd_sell.xml')
Column	payReceived	document('dvd_sell.xml')/payReceived
Column	name	document('dvd_sell.xml')/name
Table	dvd_order	document('dvd_order.xml')
Column	payDue	document('dvd_order.xml')/Due
Column	name	document('dvd_order.xml')/name

This assertion will be translated into the uniform constraint model as Example 3.4.5 shows.

Again by “UPDATE” operation, we can define also a trigger by only replacing the “INSERT” with “UPDATE” in Example 3.4.5.

All the ASSERTION constraints can be mapped in this way.

```
DECLARE NAMESPACE
  ns1=doc('dvd_sell.xml'),
  ns2=doc('dvd_order.xml')
CREATE CONSTRAINT check_ receivedEqualsDue
ON INSERT OF ns1/dvd_selling
FOR EACH NODE
Let $received:=NEW_NODE//payReceived
    $name:=NEW_NODE//dvd_name
    $due:=( FOR &due in ns2//payDue
    WHERE ns2/name=$name
          RETURN $due)
WHEN  $received>$due
DO (PopupErrorMessage("payment received date must be
       earlier than the due date"))
```

Example 3.4.5: ASSERTION in Uniform Constraint Model

3.4.2 Expression of STEP/EXPRESS Constraints

We use local "WHERE" rules in Entity definition in Example 3.4.6 to show how they are mapped into our constraint model. Other constraints from STEP/EXPRESS can be mapped in the same way.

```
ENTITY circular_cone
SUBTYPE OF(primitive_with_one_axis);
    Semi_angle: REAL;
    Radius: REAL;
    Height: REAL;
WHERE
    Semi_angle > 0;
    Semi_angle < 90;
    Radius >= 0;
    Height > 0;
 END_ENTITY;
```

Example 3.4.6: STEP/EXPRESS constraint

The schema mapping information is shown in Table 3.4:

Table 3.4 Schema Mapping of Entity circular_cone

Original Type	Original Name	Global Name
Entity	circular_cone	document('circular_cone.xml')
Element	Semi_angle	document('circular_cone.xml')/table/row/Semi_angle
Element	Radius	document('circular_cone.xml')/table/row/Radius
Element	Height	document('circular_cone.xml')/table/row/Height

The local “WHERE” rules will be translated into our constraint model in Example 3.4.7.

```
        DECLARE NAMESPACE ns1=doc('circular_cone.xml')
        CREATE CONSTRAINT  check_semi_angle
        ON INSERT OF ns1/table_1/row
        Let $sa:= ns1/table_1/row/Semi_angel
        WHEN  $sa<= 0 OR $sa>=90
        DO (PopupErrorMessage("Semi_angle should be more than 0
               and less than 90"))
```

Example 3.4.7: STEP/EXPRESS Constraint in Uniform Constraint Model

Here, by “UPDATE” operation we can define another trigger by only replacing the “INSERT” with “UPDATE” in Example 3.4.7. The radius checking and the height checking constraints can also be written in the same way.

3.4.3 Compensating Local Constraints

In a data integration system, the data sources are miscellaneous. There are those data sources where there is no constraint enforcement mechanism, nor do they have a constraint expression mechanism. We provide a way where these data sources can submit predicate-like constraints. By providing the schema mapping information predicate-like constraints can be translated into the uniform constraint model, too. Example 3.4.8 is the predicate-like constraint submitted by the data source.

```
      Employee.txt/salary > 5000EURO
```

Example 3.4.8: Predicate-Like Constraints

The schema mapping information is shown in Table 3.5.

Table 3.5 Schema Mapping of Employee.txt

Original Type	Original Name	Global Name
file	Employee.txt	document('Employee.xml')
column	salary	document('Employee.xml')/salary

The constraint in the uniform constraint model is shown in Example 3.4.9.

```
        DECPARE NAMESPACE  ns1=doc('Employee.xml')
        CREATE CONSTRAINT  check_salary
        ON INSERT OF ns1/salary
        Let $s:= ns1/salary
        WHEN  $s<= 5000
        DO (PopupErrorMessage("The salary must be more
                than 5000EURO!"))
```

Example 3.4.9: CHECK Constraint in Uniform Constraint Model

Here, by “UPDATE” operation we can define another trigger by only replacing the “INSERT” with “UPDATE” in Example 3.4.9.

3.4.4 Global Referential Constraints

In query optimization, referential constraints are used to eliminate joins. So, the contents in the action part (cascade delete or set null) are not important. Example 3.4.10 shows the way to record the referential constraints.

```
    DECLARE NAMESPACE
        ns1 = doc(Customer.xml),
        ns2 = doc(CustomerHobby.xml)
    CREATE CONSTRAINT CustomerMustExistConstraint
    ON INSERT OF ns2/CustomerHobby
    LET $newCustomer = NEW_NODE/Customer
    LET $cid := $newCustomer/C_ID
    LET $customer :=ns1/Customer/CUSTOMERRECORD
                /CUSTOMER[CUSTOMER_ID=$cid]
    WHEN not (fn:exists($customer))
    DO popuperrormessage('Customer must exist when a hobby is inserted!')
```

Example 3.4.10: Refential Constraint Example

3.4.5 Constraint Repository

Constraint repository is the general constraint storage, where the local constraints expressed in global schema, the global domain constraints and the global referential constraints are stored. The local constraints are translated into constraints in the global schema by the constraint wrapper. The global domain constraints which spread multiple data sources and the global referential constraints between the data sources are defined, for example, by the data integration system administrator.

3.5 A Semantic Query Optimizer

3.5.1 Traditional Semantic Query Optimization

Remind that there are altogether five techniques of semantic query optimization in traditional relational database technology [CGK$^+$99]:

1. Detection of empty result. If the query predicates are inconsistent with integrity constraints, the query does not have an answer.
2. Join elimination. A query may contain a join for which the result is known as a priori, hence it does not need to be evaluated. (For example, for some queries involving a join between two tables related through a referential integrity constraint).
3. Predicate elimination. If a predicate is known to be always true it can be eliminated from the query.
4. Join introduction. It may be advantageous to add a join with an additional relation, if that relation is relatively small compared with the original relations as well as highly selective. (This is even more appealing if the join attributes are indexed.)
5. Predicate introduction. A new predicate on an indexed attribute may allow for a more efficient access method. Similarly, a new predicate on a join attribute may reduce the cost of the join.

3.5.2 Architecture

Figure 3.7 shows the architecture of the semantic optimizer [LM08a].

The data sources can register their constraints to the optimizer through constraint wrappers (Step 1). The administrator of the data integration system can also add or delete global constraints (Step 2). The constraint wrappers will translate the local constraints into constraints in global schema with

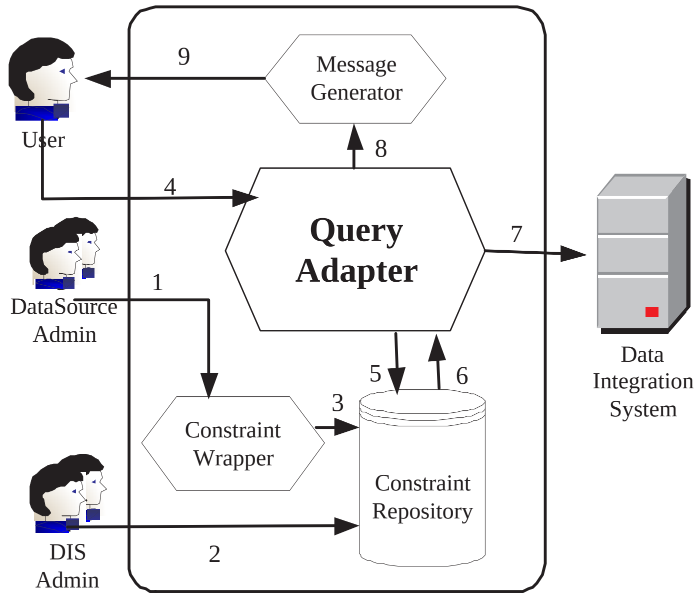


Fig. 3.7 Architecture of the Semantic Query Optimizer

the help of the schema mapping information. The translated constraints will be added into the constraint repository (Step 3). The data sources can also deregister constraints and the constraint wrappers will delete the corresponding constraints from the constraint repository.

The user can submit the query to the optimizer and the query adapter will accept the query (Step 4). Then the query adapter will consult the constraint repository (Step 5) and check whether there are constraints related to the query (Step 6). If the query adapter finds that there are some applicable constraints in the constraint repository, the constraints will be fetched out and compared with the query condition. When the query adapter finds that there is conflict between the query condition and the constraints so that the query will return empty result, a message will be generated (Step 8) and the user will be informed (Step 9). When there is no conflict, the query adapter will try to find whether there is possibility to optimize the query using the constraint information. If there exists the possibility, the query adapter will generate a new query and send it to the query processor of the data integration system (Step 7).

The semantic query optimizer is an independent module, which can either be installed as a supplement of the Data Integration System, or be published as a Web Service, whose interface can be published and invoked by different Data Integration Systems.

3.5.3 Query Adapter

Query adapter is the kernel of the semantic query optimizer. It consists of four components: query decomposer, constraint fetcher, conflict detector, and query reformulator, as illustrated in the Figure 3.8. Generally, the query adapter accepts an XML query from the user as input. Then it uses the semantic knowledge stored in the constraint repository to generate a semantically equivalent but more efficient query. The functionalities of the four components are discussed below.

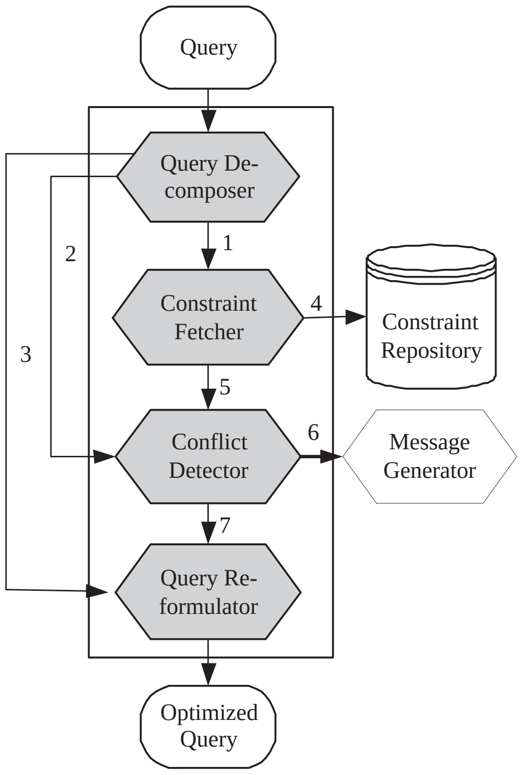


Fig. 3.8 Components and Processing Flow of Query Adapter

3.5.3.1 Query Decomposer

The query decomposer takes an XML Query from the user as input and decomposes the query into three parts: condition part, data source part and result return part. It forwards the query condition part and the data source part to constraint fetcher (Step 1), the query condition part to conflict detector (Step 2), and the whole query to query reformulator (Step 3).

3.5.3.2 Constraint Fetcher

The constraint fetcher searches the constraint repository to find the related constraints (Step 4) and forwards the constraints to conflict detector (Step 5).

3.5.3.3 Conflict Detector

The conflict detector takes query condition and the constraints as input. We suppose that there are no contradictory constraints in the constraint repository (e.g., $a > 10 \wedge a < 9$ is a contradictory constraint). Since the constraints and the query condition can be arbitrary literals, we use the following method. The query condition part is transformed into DNF (Disjunctive Normal Form) [Men97]. The constraints are transformed into DNF, too. Converting a logical expression to DNF involves using Double Negation Elimination, De Morgan's Law and the Distributive Law [Sch00]. Conflict detector builds a constraint DNF tree. Then it uses the constraint DNF tree to evaluate the query condition. If there is a conflict, it will return false. The message generator is informed and generates an error message (Step 6). If there is no conflict, query reformulator will be invoked (Step 7). We use an example to show how this algorithm works.

Suppose the constraints are:

$P_d = (((a_1 > 5) \vee (a_2 > 30)) \wedge ((a_2 < 50) \vee (a_3 > 40)))$

Suppose the condition in the query is:

$P_c = (((a_1 > 10) \vee (a_2 < 30)) \wedge (a_1 < 20)))$

The whole element set for the constraint and the query condition is:

$A_t = \{a_1, a_2, a_3\}$

The constraint will be transformed into the following DNF:

$P_d = (((a_1 > 5) \wedge (a_2 < 50)) \vee ((a_1 > 5) \wedge (a_3 > 40)) \vee ((a_2 > 30) \wedge (a_2 < 50)) \vee ((a_2 > 30) \wedge (a_3 > 40)))$

The query condition is also transformed into DNF:

$P_c = (((a_1 > 10) \wedge (a_1 < 20)) \vee ((a_1 < 20) \wedge (a_2 < 30)))$

Both DNFs will be transformed into DNF trees. The root is the "$\vee$" operator and the leaves are the conjunctions "$\wedge$".

Each leaf of the query condition DNF tree will be evaluated with each leaf of the constraint DNF tree. When there is one leaf in condition DNF tree which is evaluated to be true with one leaf in the constraint DNF tree, the final result is true. Otherwise it is evaluated to be false.

In this example, the query condition predicate $((a_1 > 10) \wedge (a_2 < 20))$ and the constraint predicate $((a_1 > 5) \wedge (a_2 < 50))$ are evaluated to be true, hence, there is no conflict between the constraint and the query.

3.5.3.4 Query Reformulater

If the conflict detector finds no conflict, the query reformulator will check whether there exist possibilities to eliminate joins or predicates. If this is the case, it will generate a new query, which is semantically equivalent to and hopefully more efficient than the original one. Query reformulation rules are discussed in Section 3.5.4.

3.5.4 Query Reformulation Rules

3.5.4.1 Analysis

There are five semantic query optimization techniques most often discussed in literature: join elimination, join introduction, predicate elimination, predicate introduction, and detection of empty results [CGK^{+}99]. Among the five techniques, join introduction and predicate introduction are based on the index mechanism of a relational database. Due to the fact that in an XML-based data integration system the data sources are often non-relational and most of them do not have an indexing mechanism, these two techniques are not considered. Join elimination, predicate elimination and detection of empty results are used as the premier query reformulation rules in our optimizer.

3.5.4.2 Detection of Empty Results

When the conflict detector detects that there is a conflict between constraints and the query so that the query will return an empty result set, the message generator will inform the user about the detection result.

3.5.4.3 Predicate Elimination

The main idea of predicate elimination is that if a predicate is known to be true, it can be eliminated from the query. For example, suppose there is a constraint: $Customer\_salary > 50000$ and there is a query in which there is a condition $customer\_salary > 40000$. This condition predicate can be eliminated.

Still, we use the constraint source DNF tree and the query condition DNF tree to test whether there exists some possibility to eliminate predicates. If the domain predicate of query condition is subsumed as true by each leaf in the constraint DNF tree, we can eliminate the domain predicate from the query condition. We use an example to explain the whole precess.

Assume that a query contains the condition that:

$((A > 10) \wedge (B > 20))$

Assume that DNF tree constructed by all associated constraints is:

$(((A > 20) \wedge (B > 25)) \vee ((A > 30) \wedge (B < 50)))$

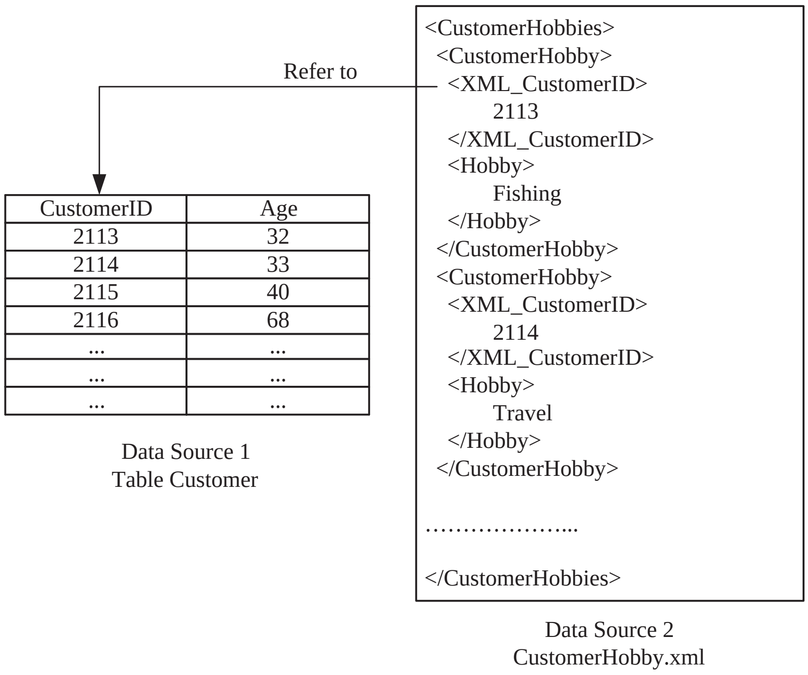
It is clear, the literal $(A > 10)$ subsumes $(A > 20) and (A > 30)$. In this way, $(A > 10)$ can be removed from the query condition. The literal $(B > 20)$ subsumes $(B > 25)$, however, does not subsume $(B < 50)$. Finally, the literal $(B > 20)$ stays.

3.5.4.4 Join Elimination

The main idea of join elimination is that when a query contains a join for which the result is known as a priori, it does not need to be evaluated. For example, when the two attributes of the join are related by an enforced referential integrity constraint.

Referential constraints are mainly used here. The query transformer takes the selected referential constraints and the query condition as input to see whether there exist redundant joins. The first step is to extract the attributes in the query condition part. Then the algorithm will find all the data sources that have these attributes. The algorithm will search the constraints which are filtered out according to the query condition and result part. When there are referential constraints, the algorithm will build a reference chain, where the father source (which can be a table from a relational database or an XML file from an XML data source) is the source, the primary key belongs to, and the child source is the data source, the foreign key belongs to. The algorithm will search the reference chain and extract all the attributes from the query return part. If there is no attribute from the first data source in the query return part and there is no attribute from the first data source in the query condition part except the primary key in the join, the join between the primary key and the foreign key will be eliminated. We also use an example to explain this reformulation rule.

Suppose there is a relational data source $DS1$. Suppose in $DS1$ table $Customer$ has a primary key $CustomerID$. Table $Customer$ contains two columns: $CustomerID$ and Age. Suppose there is an XML data source $DS2$. In $DS2$ there is an XML file $CustomerHobby.xml$. The file $CustomerHobby.xml$ contains two elements: $XML\_CustomerID$ and $Hobby$. $CustomerID$ in the table $Customer$ is referenced as the foreign key of $CustomerHobby$. This global referential constraint is defined and stored in the constraint repository. After integration the table $Customer$ will be seen by the data integration system user as $Customer.xml$. Suppose the primary key and the foreign key are not null. Figure 3.9 is used as an example to explain this procedure.



CustomerID	Age
2113	32
2114	33
2115	40
2116	68
...	...
...	...
...	...

```
<CustomerHobbies>
 <CustomerHobby>
  <XML_CustomerID>
      2113
  </XML_CustomerID>
  <Hobby>
      Fishing
  </Hobby>
 </CustomerHobby>
 <CustomerHobby>
  <XML_CustomerID>
      2114
  </XML_CustomerID>
  <Hobby>
      Travel
  </Hobby>
 </CustomerHobby>

………………...

</CustomerHobbies>
```

Fig. 3.9 Example to Explain Join Elimination

Suppose there is a query, which wants to find all values of $Hobby$, where $XML\_CustomerID$ is greater than 2113 and $CustomerID$ is joined with $XML\_CustomerID$. Formally, the query condition is:

$XML\_CustomerID > 2113 \wedge XML\_CustomerID = CustomerID$

There is a join in the query condition. After the constraint fetcher queries the constraint repository, a referential constraint is found. According to the referential constraint, we build the following reference chain:

$Customer \leftarrow CustomerHobby$

We use $\leftarrow$ to mean "being referenced". In this reference chain the first data source is $Customer$. Then the algorithm will find whether there exist elements from $Customer$ in the query return part. In this example there is only one element *hobby* in the query return part, which is not from $Customer$. Then the algorithm will analyze the elements in the query condition part. If the elements in the query condition part are not from $Customer$ except the primary key in the join, the algorithm will eliminate this join in the condition part. As a result, the data integration system does not need to retrieve both data sources. This saves execution and communication cost.

3.5.5 Experiments and Evaluation

We used BEA Weblogic as the application server, BEA LiquidData as the data integration system [BEA03], Tamino XML Server [AG06] as constraint repository, Apache Tomcat [Fou06] to run the web service. We used DB2 as relational data source and XML files as a non-relational data source to carry out our experiments. Five data sources are used to participate in the whole experiments:

1. A DB2 database named CSCO, which contains customer information in the table CUSTOMER and the corresponding order information in the table CUSTOMER_ORDER, referenced using the attribute CUSTOMER_ID;
2. A DB2 database named CSC, which contains the data replication of the customer information, i.e., the CUSTOMER table in CSCO;
3. A DB2 database named CSO, which contains the data replication of the customer order information, i.e., the CUSTOMER_ORDER table in CSCO. We define a global referential constraint between these two data sources: CSC and CSO. The CUSTOMER_ID in CUSTOMER from CSC is referenced by CUSTOMER_ORDER from CSO.
4. An XML file named xml_customer.xml, which contains customer information. There is a global referential constraint defined between CSO and xml_customer.xml: The attribute CSO.CUSTOMER_ORDER.CUSTOMER_ID references the element in xml_customer.CUSTOMER.CUSTOMER_ID.
5. An XML file named xml_customer_order.xml, which contains customer order information. Again we define a global referential constraint between the two XML data sources: the CUSTOMER_ID in xml_customer.xml is referenced by xml_customer_order.xml. We define another global referential constraint between CSC and xml_customer_order.xml: The element xml_customer_order.CUSTOMER_ORDER.CUSTOMER_ID references the attribute CSC.CUSTOMER.CUSTOMER_ID.

For these experiments, several data schemata are designed for the data sources. The data schemata for DB2 database are represented as SQL in Example 3.5.1 (Table CUSTOMER) and Example 3.5.2 (Table CUSTOMER_Order).

```
CREATE TABLE CUSTOMER (
CUSTOMER_ID DECIMAL(8) PRIMARY KEY NOT NULL,
FIRST_NAME   VARCHAR(20) NOT NULL,
LAST_NAME    VARCHAR(20) NOT NULL,
CUSTOMER_SINCE DATE NOT NULL,
STREET_ADDRESS1 VARCHAR(50) NOT NULL,
STREET_ADDRESS2 VARCHAR(50),
CITY VARCHAR(40) NOT NULL,
STATE VARCHAR(40) NOT NULL,
ZIPCODE VARCHAR(10) NOT NULL,
EMAIL_ADDRESS VARCHAR(40) NOT NULL,
TELEPHONE_NUMBER VARCHAR(20) NOT NULL);
```

Example 3.5.1: Table CUSTOMER

```
CREATE TABLE CUSTOMER_ORDER (
ORDER_ID DECIMAL(8) PRIMARY KEY NOT NULL,
ORDER_DATE DATE NOT NULL,
CUSTOMER_ID DECIMAL(8) NOT NULL,
SHIP_METHOD VARCHAR(20) NOT NULL,
TOTAL_ORDER_AMOUNT DECIMAL(10,2) NOT NULL);
```

Example 3.5.2: Table CUSTOMER_Order

The data schemata for XML files as data sources are represented as XML SCHEMA in Example 3.5.3 (xml_Customer.xml) and in Example 3.5.4 (xml_Customer_order.xml).

Consider that there is an integrity constraint submitted by the local database CSCO administrator, written by using the expression of CHECK CONSTRAINT, as shown in Example 3.5.5. It indicates that the value of attribute TOTAL_ORDER_AMOUNT is between 300 and 3000. This constraints will be expressed in the uniform constraint as shown in Example 3.5.6.

There is a referential integrity constraint submitted by the local database CSCO administrator, shown in Example 3.5.7.

There is also a submitted integrity constraint (IC3) in the local database CSC in Example 3.5.8. IC3 will be translated by the constraint wrapper in the following uniform constraint model as shown in Example 3.5.9.

A referential constraint between two data sources (CSC and CSO) is defined in IC4, as shown in Example 3.5.10.

A referential constraint between two XML files (xml_customer.xml and xml_customer_order.xml) is defined in IC5, as shown in Example 3.5.11.

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
    elementFormDefault="qualified">
  <xs:element name="db">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="CUSTOMER" maxOccurs="unbounded"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:element name="CUSTOMER">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="FIRST_NAME"/>
        <xs:element ref="LAST_NAME"/>
        <xs:element ref="CUSTOMER_ID"/>
        <xs:element ref="STATE"/>
        <xs:element ref="ZIPCODE"/>
        <xs:element ref="CITY"/>
        <xs:element ref="STREET_ADDRESS2"/>
        <xs:element ref="STREET_ADDRESS1"/>
        <xs:element ref="CUSTOMER_SINCE"/>
        <xs:element ref="EMAIL_ADDRESS"/>
        <xs:element ref="TELEPHONE_NUMBER"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:element name="CUSTOMER_ID" type="xs:long"/>
  <xs:element name="CUSTOMER_SINCE" type="xs:string"/>
  <xs:element name="EMAIL_ADDRESS" type="xs:string"/>
  <xs:element name="FIRST_NAME" type="xs:string"/>
  <xs:element name="LAST_NAME" type="xs:string"/>
  <xs:element name="STATE" type="xs:string"/>
  <xs:element name="STREET_ADDRESS1" type="xs:string"/>
  <xs:element name="STREET_ADDRESS2" type="xs:string"/>
  <xs:element name="TELEPHONE_NUMBER" type="xs:string"/>
  <xs:element name="ZIPCODE" type="xs:string"/>
  <xs:element name="CITY" type="xs:string"/>
</xs:schema>
```

Example 3.5.3: XML Schema of xml_Customer.xml

Example 3.5.12 is a constraint in the XML file data sources. It indicates that the value of attribute TOTOAL_ORDER_AMOUNT in the xml_customer_order.xml is between 3300 and 6000. Otherwise an error message will appear and the insert operation will be discarded.

We define a global referential constraint between the relational database CSO and the XML file xml_customer.xml, shown in Example 3.5.13. It indicates that the attribute CUSTOMER_ORDER.CUSTOMER_ID in the CSO references the attribute CUSTOMER.CUSTOMER_ID in the xml_customer.

We define another global referential cosntraint between the relational database CSC and the XML file xml_customer_order.xml as shown in Exam-

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
    elementFormDefault="qualified">
  <xs:element name="db">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="CUSTOMER_ORDER"
          maxOccurs="unbounded"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:element name="CUSTOMER_ID" type="xs:long"/>
  <xs:element name="CUSTOMER_ORDER">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="ORDER_DATE"/>
        <xs:element ref="ORDER_ID"/>
        <xs:element ref="CUSTOMER_ID"/>
        <xs:element ref="SHIP_METHOD"/>
        <xs:element ref="TOTAL_ORDER_AMOUNT"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:element name="ORDER_DATE" type="xs:string"/>
  <xs:element name="ORDER_ID" type="xs:long"/>
  <xs:element name="SHIP_METHOD" type="xs:string"/>
  <xs:element name="TOTAL_ORDER_AMOUNT"
     type="xs:decimal"/>
</xs:schema>
```

Example 3.5.4: XML Schema of xml_Customer_order.xml

```
IC1:
CONSTRAINT constraint1
CHECK ((CSCO.CUSTOMER_ORDER.TOTAL_ORDER_AMOUNT >= 300)
  and (CSCO.CUSTOMER_ORDER.TOTAL_ORDER_AMOUNT <= 3000))
```

Example 3.5.5: IC1: CHECK constraint in Table CUSTOMER_ORDER of CSCO Database

```
IC1:
        CREATE CONSTRAINT IC1
        ON INSERT OF doc('CSCO.xml')/db/CUSTOMER_ORDER
        LET $c1 :=
          doc('CSCO.xml')/db/CUSTOMER_ORDER/TOTAL_ORDER_AMOUNT
        WHEN (($c1 < 300) or ($c1 > 3000))
        DO (PopupErrorMessage("Column cannot be inserted!
           Total Order Amount shoule be between 300 and 3000!"))
```

Example 3.5.6: IC1 in Uniform Constraint Model

```
IC2:
The attribute CSCO.CUSTOMER_ORDER.CUSTOMER_ID
references the attribute CSCO.CUSTOMER.CUSTOMER_ID.
```

Example 3.5.7: Referential Constraint between CUSTOMER and CUSTOMER_ORDER Tables

```
    IC3:
    CONSTRAINT constraint2
    CHECK ((CSO.CUSTOMER_ORDER.TOTAL_ORDER_AMOUNT >= 300)
    and (CSO.CUSTOMER_ORDER.TOTAL_ORDER_AMOUNT <= 3000))
```

Example 3.5.8: IC3: CHECK Constraint in CSO Database

```
    IC3:
        CREATE CONSTRAINT IC1
        ON INSERT OF doc('CSO.xml')/db/CUSTOMER_ORDER
        LET $c1 :=
          doc('CSO.xml')/db/CUSTOMER_ORDER/TOTAL_ORDER_AMOUNT
        WHEN ($c1 < 300)
        DO (PopupErrorMessage("Column cannot be inserted!
         Total Order Amount shoule be greater than or equal to 300!"))
```

Example 3.5.9: IC3 in Uniform Constraint Model

```
    IC4:
    The attribute CSO.CUSTOMER_ORDER.CUSTOMER_ID
    references the attribute CSC.CUSTOMER.CUSTOMER_ID.
```

Example 3.5.10: IC4: Referential Constraint between Two Data Sources

```
    IC5:
    The attribute xml_customer.xml/CUSTOMER_1/CUSTOMER_ID
    references
    the attribute xml_customer_order.xml/CUSTOMER_ORDER2/CUSTOMER_ID.
```

Example 3.5.11: IC5: Referential Constraint between Two XML Files

```
    IC6:
    CREATE CONSTRAINT trigger1
    ON
    INSERT OF doc('xml_customer_order.xml')
       /db/CUSTOMER_ORDER
    LET $c1 :=
    doc('xml_customer_order.xml')
       /db/CUSTOMER_ORDER
             /TOTAL_ORDER_AMOUNT
    WHEN ($c1 < 3300)
    DO (PopupErrorMessage
            ("Column cannot be inserted"))
```

Example 3.5.12: IC6: XML File Constraint in Uniform Constraint Model

```
    IC7:
        The attribute CSO.CUSTOMER_ORDER.CUSTOMER_ID
            references
        the element in xml\_customer.CUSTOMER.CUSTOMER_ID.
```

Example 3.5.13: IC7: A referential Constraint between Database and XML File ple 3.5.14. IC7 indicates that the element CUSTOMER_ID in xml_customer_order references the attribute CUSTOMER.CUSTOMER_ID in CSC.

```
IC8:
The element xml\_customer\_order.CUSTOMER_ORDER.CUSTOMER_ID
references the attribute CSC.CUSTOMER.CUSTOMER_ID.
```

Example 3.5.14: IC8: Referential Constraint betweem DB and XML File

3.5.5.1 Notations in Performance Table

We use the notations given in Table 3.6 to explain the performance numbers shown in the next paragraphs.

Table 3.6 Notations in Performance Table

Q	Query
DV	Data Volume
OrT	Original Execution Time (s)
TrT	Transformation Time (s)
OpT	Optimized Execution Time (s)
AB	Absolute Benefit (s)
RB	Relative Benefit (%)
DVR	Data Volume of Relational Data
DVX	Data Volume of XML Data
OrDRR	Original Data Retrieval Time of Relational Data(s)
OrDRX	Original Data Retrieval Time of XML Data(s)
OpDRR	Optimized Data Retrieval Time of Relational Data(s)
OpDRX	Optimized Data Retrieval Time of XML Data(s)

Note that our example queries are tested in BEA LiquidData. When "#" appears in the query examples in the following paragraphs, it means an input parameter in LiquidData.

3.5.5.2 Detection of Empty Results

We design two queries, $Q1$ and $Q2$, which are built to return empty result by our optimizer. $Q1$ is a simple selection query over only the relational database CSCO, while $Q2$ is the same query over only the XML file xml_customer_order.xml. In Q1 (Example 3.5.15), only one DB2 database named CSCO was used as data source. In Q2 (Example 3.5.16), only an XML file named xml_customer_order.xml was used as data source.

```
Example Q1:
<results>
{
  for $CSCO.CUSTOMER_ORDER_1 in document("CSCO")
     /db/CUSTOMER_ORDER
  where ($CSCO.CUSTOMER_ORDER_1/TOTAL_ORDER_AMOUNT ge
  $#min_total_order_amount of type xs:decimal)
  return
  <result>
    <order_id>
      {xf:data($CSCO.CUSTOMER_ORDER_1/ORDER_ID)}
    </order_id>
    <order_date>
      {xf:data($CSCO.CUSTOMER_ORDER_1/ORDER_DATE)}
    </order_date >
    <ship_method>
      {xf:data($CSCO.CUSTOMER_ORDER_1/SHIP_METHOD)}
    </ship_method>
  </result>
}
</results>
```

Example 3.5.15: Q1

```
Example Q2:
<results>
{
  for $xml_customer_order.CUSTOMER_ORDER_1 in
  document("xml_customer_order")/db/CUSTOMER_ORDER
  where ($xml_customer_order.CUSTOMER_ORDER_1
      /TOTAL_ORDER_AMOUNT ge
  $#min_total_order_amount of type xs:decimal)
  return
  <result>
    <order_id>
      {xf:data($xml_customer_order.CUSTOMER_ORDER_1/ORDER_ID)}
    </order_id>
    <order_date>
      {xf:data($xml_customer_order.CUSTOMER_ORDER_1/ORDER_DATE)}
    </order_date>
    <ship_method>
      {xf:data($xml_customer_order.CUSTOMER_ORDER_1/SHIP_METHOD)}
    </ship_method>
  </result>
}
</results>
```

Example 3.5.16: Q2

After optimization both XQueries are transformed to empty ones because of the constraints. That means that the results of these XQueries should be empty according to the constraints stored in the constraint repository.

We use 1000, 5000 and 15000 records respectively in the data sources for testing and get the performance results, as shown in Table 3.7. From Table 3.7 we can see that great benefit can be obtained by applying detection of empty result technique. Another fact we find is that the larger the data volume is, the more improvement we obtain.

Table 3.7 Performance of Detection of Empty Results

Q	DV	OrT(s)	TrT(s)	OpT(s)	AB(s)	RB	Constraint Used
Q1	1000	0.509	0.047	-	0.462	90.77%	IC1
	5000	1.112	0.047	-	1.065	95.77%	
	15000	2.234	0.047	-	2.187	97.90 %	
Q2	1000	0.875	0.047	-	0.823	94.06%	IC3
	5000	3.906	0.047	-	3.854	98.67%	
	15000	12.469	0.047	-	12.417	99.58 %	

3.5.5.3 Join Elimination

We design three queries: $Q3$, $Q4$ and $Q5$, whose joins are eliminated by our optimizer. $Q3$ (Example 3.5.17) is queried only over the relational database CSCO, $Q4$ (Example 3.5.18) is queried over two relational databases, CSC and CSO, and $Q5$ (Example 3.5.19) is queried over two XML files: xml_customer.xml and xml_customer_order.xml.

After join elimination optimization these three XQueries are transformed to new ones, which may be executed more efficient. $Q3$ is rewritten to $Q3'$(Example 3.5.20), $Q4$ is rewritten to $Q4'$ (Example 3.5.21) and $Q5$ is rewritten to $Q5'$ (Example 3.5.22).

We use 1000, 5000, and 15000 records respectively in the data sources for testing. We get the performance results, as shown in Table 3.8. From Table 3.8 we can see that our optimizer is capable to efficiently optimize queries. Again we find the fact that the larger the data volume is, the more improvement we obtain.

```
Example Q3:
<results>
{
  for $CSCO.CUSTOMER_ORDER_1 in
     document("CSCO")/db/CUSTOMER_ORDER
  for $CSCO.CUSTOMER_2 in
     document("CSCO")/db/CUSTOMER
  where ($CSCO.CUSTOMER_2/CUSTOMER_ID eq
  $CSCO.CUSTOMER_ORDER_1/CUSTOMER_ID)
   and ($CSCO.CUSTOMER_2/CUSTOMER_ID
  ge $#min_customer_id of type xs:decimal)
  return
  <result>
    <order_id>
      {xf:data($CSCO.CUSTOMER_ORDER_1/ORDER_ID)}
    </order_id>
    <order_date>
      {xf:data($CSCO.CUSTOMER_ORDER_1/ORDER_DATE)}
    </order_date>
    <ship_method>
      {xf:data($CSCO.CUSTOMER_ORDER_1/SHIP_METHOD)}
    </ship_method>
  </result>
}
</results>
```

Example 3.5.17: Q3

```
Example Q4:
<results>
{
for $CSC.CUSTOMER_1
  in document("CSC")/db/CUSTOMER
for $CSO.CUSTOMER_ORDER_2
  in document("CSO")/db/CUSTOMER_ORDER
where ($CSC.CUSTOMER_1/CUSTOMER_ID
  eq $CSO.CUSTOMER_ORDER_2/CUSTOMER_ID)
and ($CSC.CUSTOMER_1/CUSTOMER_ID
  ge $#min_customer_id of type xs:decimal)
return
  <result>
    <order_id>
      {xf:data($CSO.CUSTOMER_ORDER_2/ORDER_ID)}
    </order_id>
    <order_date>
      {xf:data($CSO.CUSTOMER_ORDER_2/ORDER_DATE)}
    </order_date>
    <ship_method>
      {xf:data($CSO.CUSTOMER_ORDER_2/SHIP_METHOD)}
    </ship_method>
  </result>
}
</results>
```

Example 3.5.18: Q4

```
Example Q5:
<results>
{
for $xml_customer.CUSTOMER_1
  in document("xml_customer")/db/CUSTOMER
for $xml_customer_order.CUSTOMER_ORDER_2 in
document("xml_customer_order")/db/CUSTOMER_ORDER
where ($xml_customer.CUSTOMER_1/CUSTOMER_ID eq
$xml_customer_order.CUSTOMER_ORDER_2/CUSTOMER_ID) and
($xml_customer.CUSTOMER_1/CUSTOMER_ID
  ge $#min_customer_id of type
xs:decimal)
return
  <result>
    <order_id>
      {xf:data($xml_customer_order.CUSTOMER_ORDER_2/ORDER_ID)}
    </order_id>
    <order_date>
      {xf:data($xml_customer_order.CUSTOMER_ORDER_2/ORDER_DATE)}
    </order_date>
    <ship_method>
      {xf:data($xml_customer_order.CUSTOMER_ORDER_2/SHIP_METHOD)}
    </ship_method>
  </result>
}
</results>
```

Example 3.5.19: Q5

```
Q3':
<results>
{
  for $CSCO.CUSTOMER_ORDER_1
     in document("CSCO")/db/CUSTOMER_ORDER
  where ($CSCO. CUSTOMER_ORDER_1/CUSTOMER_ID
   ge $#min_customer_id of type
  xs:decimal)
  return
  <result>
    <order_id>
      {xf:data($CSCO.CUSTOMER_ORDER_1/ORDER_ID)}
    </order_id>
    <order_date>
      {xf:data($CSCO.CUSTOMER_ORDER_1/ORDER_DATE)}
    </order_date>
    <ship_method>
      {xf:data($CSCO.CUSTOMER_ORDER_1/SHIP_METHOD)}
    </ship_method>
  </result>
}
</results>
```

Example 3.5.20: Q3'

```
Q4':
<results>
{
  for $CSO.CUSTOMER_ORDER_2
    in document("CSO")/db/CUSTOMER_ORDER
  where ($CSO.CUSTOMER_ORDER_2/CUSTOMER_ID
    ge $#min_customer_id of type
  xs:decimal)
  return
  <result>
    <order_id>
      {xf:data($CSO.CUSTOMER_ORDER_2/ORDER_ID)}
    </order_id>
    <order_date>
      {xf:data($CSO.CUSTOMER_ORDER_2/ORDER_DATE)}
    </order_date>
    <ship_method>
      {xf:data($CSO.CUSTOMER_ORDER_2/SHIP_METHOD)}
    </ship_method>
  </result>
}
</results>
```

Example 3.5.21: Q4'

```
Q5':
<results>
{
  for $xml_customer_order.CUSTOMER_ORDER_2 in
     document("xml_customer_order")/db/CUSTOMER_ORDER
  where ($xml_customer_order.CUSTOMER_ORDER_2/CUSTOMER_ID ge
  $#min_customer_id of type xs:decimal)
  return
  <result>
    <order_id>
      {xf:data($xml_customer_order.CUSTOMER_ORDER_2/ORDER_ID)}
    </order_id>
    <order_date>
      {xf:data($xml_customer_order.CUSTOMER_ORDER_2/ORDER_DATE)}
    </order_date>
    <ship_method>
      {xf:data($xml_customer_order.CUSTOMER_ORDER_2/SHIP_METHOD)}
    </ship_method>
  </result>
}
</results>
```

Example 3.5.22: Q5'

Table 3.8 Performance of Join Elimination

Q	DV	OrT(s)	TrT(s)	OpT(s)	RB	Constraint Used
Q3	1000	0.922	0.141	0.314	50.65%	IC2
	5000	1.982	0.141	0823	51.36%	
	15000	5.712	0.141	2.609	51.86 %	
Q4	1000	1.687	0.141	0.272	75.52%	IC4
	5000	4.489	0.141	0.92	76.36%	
	15000	11.953	0.141	2.588	77.17 %	
Q5	1000	1.86	0.153	0.797	48.92%	IC5
	5000	7.672	0.153	3.714	49.60%	
	15000	23.266	0.153	11.487	49.97 %	

For the queries whose data sources are heterogeneous, for example, one is relational database, another is an XML file, we use the two examples Q6 (shown in Example 3.5.23) and Q7 (shown in Example 3.5.24) for testing.

```
Q6:
<results>
{
for $xml_customer.CUSTOMER_1
   in document("xml_customer")/db/CUSTOMER
for $xml_customer_order.CUSTOMER_ORDER_2 in
        document("CSO")/db/CUSTOMER_ORDER
where ($xml_customer.CUSTOMER_1/CUSTOMER_ID eq
        $xml_customer_order.CUSTOMER_ORDER_2/CUSTOMER_ID) and
            ($xml_customer.CUSTOMER_1/CUSTOMER_ID lt 10)
return
    <result>
     <order_id>
      {xf:data($xml_customer_order.CUSTOMER_ORDER_2/ORDER_ID)}
     </order_id>
     <order_date>
         {xf:data($xml_customer_order.CUSTOMER_ORDER_2/ORDER_DATE)}
     </order_date>
     <ship_method>
        {xf:data($xml_customer_order.CUSTOMER_ORDER_2/SHIP_METHOD)}
        </ship_method>
  </result>
}
</results>
```

Example 3.5.23: Q6

Because of the $IC6$, the join "(xml_customer.CUSTOMER_1/CUSTOMER_ID *eq* xml_customer_order.CUSTOMER_ORDER_2/CUSTOMER_ID)" will be eliminated by our optimizer and $Q6$ will be semantically equivalent to $Q6'$ Example 3.5.25.

```
<results>
{
for $xml_customer.CUSTOMER_1 in document("CSC")/db/CUSTOMER
for $xml_customer_order.CUSTOMER_ORDER_2 in
    document("xml_customer_order")/db/CUSTOMER_ORDER
where ($xml_customer.CUSTOMER_1/CUSTOMER_ID eq
    $xml_customer_order.CUSTOMER_ORDER_2/CUSTOMER_ID) and
        ($xml_customer.CUSTOMER_1/CUSTOMER_ID lt 10)
return
  <result>
    <order_id>
      {xf:data($xml_customer_order.CUSTOMER_ORDER_2/ORDER_ID)}
    </order_id>
    <order_date>
      {xf:data($xml_customer_order.CUSTOMER_ORDER_2/ORDER_DATE)}
    </order_date>
    <ship_method>
      {xf:data($xml_customer_order.CUSTOMER_ORDER_2/SHIP_METHOD)}
    </ship_method>
  </result>
}
</results>
```

Example 3.5.24: Q7

```
Q6':
<results> {
    for $xml_customer_order.CUSTOMER_ORDER_2 in
        document('CSO')/db/CUSTOMER_ORDER
    where ($xml_customer_order.CUSTOMER_ORDER_2/CUSTOMER_ID lt 10)
return
    <result>
     <order_id>
      {xf:data($xml_customer_order.CUSTOMER_ORDER_2/ORDER_ID)}
     </order_id>
     <order_date>
      {xf:data($xml_customer_order.CUSTOMER_ORDER_2/ORDER_DATE)}
     </order_date>
     <ship_method>
       {xf:data($xml_customer_order.CUSTOMER_ORDER_2/SHIP_METHOD)}
     </ship_method>
    </result> }
</results>
```

Example 3.5.25: Q6'

Q7 is optimized and the join is eliminated and there is no need to access the relational database CSC, shown in Example 3.5.26.

Again we use different data volume combinations to test our performance. The combination includes:

1. Small data volume RDB with small data volume XML file, RDB is eliminated.

```
<results> {
for $xml_customer_order.CUSTOMER_ORDER_2 in
    document('xml_customer_order')/db/CUSTOMER_ORDER
where ($xml_customer_order.CUSTOMER_ORDER_2/CUSTOMER_ID lt 10)
return
 <result>
  <order_id>
   {xf:data($xml_customer_order.CUSTOMER_ORDER_2/ORDER_ID)}
  </order_id>
  <order_date>
   {xf:data($xml_customer_order.CUSTOMER_ORDER_2/ORDER_DATE)}
  </order_date>
  <ship_method>
   {xf:data($xml_customer_order.CUSTOMER_ORDER_2/SHIP_METHOD)}
  </ship_method>
 </result> }
</results>
```

Example 3.5.26: Q7'

2. Small data volume RDB with huge data volume XML file, RDB is eliminated.
3. Small data volume RDB with small data volume XML file, XML file is eliminated.
4. Small data volume RDB with huge data volume XML file, XML file is eliminated.
5. Huge data volume RDB with small data volume XML file, RDB is eliminated.
6. Huge data volume RDB with small data volume XML file, XML file is eliminated.
7. Huge data volume RDB with huge data volume XML file, RDB is eliminated.
8. Huge data volume RDB with Huge data volume XML file, XML file is eliminated.

We test two kinds of situations: one is that the return result set is small (under 50 record sets), the other is that the return result set is huge (e.g., about 10000 record sets). The performance with small result set is shown in Table 3.9. The data retrieval time before and after optimization when the result set is small is shown in Table 3.10.

Table 3.9 Performance of Join Elimination with DB2 and XML file when the Result Set is Small

Q	DVR	DVX	OrT(s)	TrT(s)	OpT(s)	AB(s)	RB	Constraint Used
Q6	1000	1000	11.219	0.875	0.172	10.172	90.67%	IC7
	15000	1000	12.343	0.875	0.14	11.328	91.78 %	
	15000	15000	23.468	0.875	0.11	22.483	95.80 %	
	1000	15000	41.094	0.875	0.75	39.469	96.05 %	
Q7	1000	1000	13.797	1.938	3.566	8.343	60.47%	IC8
	15000	1000	25.281	1.938	38.093	-14.75	-58.34 %	
	15000	15000	204.688	1.938	19.125	183.526	89.71 %	
	1000	15000	103.453	1.938	12.203	89.312	86.33 %	

Table 3.10 Comparison of Data Retrieval Time before and after Join Elimination with DB2 and XML file when the Result Set is Small

Q	DVR	DVX	OrDRR	OrDRX	OpDRR	OpDRX
Q6	1000	1000	2.516	7.047	0.172	-
	15000	1000	2.641	6.812	0.125	-
	15000	15000	0.485	17.844	0.094	-
	1000	15000	2.454	31.765	0.719	-
Q7	1000	1000	4.378	7.218	-	1.844
	15000	1000	11.656	10.063	-	11.968
	15000	15000	61.109	136.641	-	12.922
	1000	15000	35.677	62.734	-	6.093

The performance with large result set is shown in Table 3.11. The data retrieval time before and after optimization when the result set is huge is shown in Table 3.12.

Table 3.11 Performance of Join Elimination with DB2 and XML file when the Result Set is Huge

Q	DVR	DVX	OrT(s)	TrT(s)	OpT(s)	AB(s)	RB
Q6	1000	1000	13.578	0.875	5.062	7.641	56.27%
	15000	1000	13.719	0.875	10.172	2.672	19.48 %
	15000	15000	51.562	0.875	4.578	46.109	89.42 %
	1000	15000	33.422	0.875	1.187	31.36	93.83 %
Q7	1000	1000	5.719	1.938	3.875	-0.094	-1.64%
	15000	1000	3.984	1.938	2.984	-0.938	-23.54 %
	15000	15000	23.719	1.938	23.11	-1.329	-5.6 %
	1000	15000	152.453	1.938	37.016	113.499	74.45 %

From Table 3.9 and Table 3.11 we can see that the join elimination does not always lead to performance improvement. Especially when the access to the relational database is eliminated, join elimination even leads to a subopti-

Table 3.12 Comparison of Data Retrieval Time before and after Join Elimination with DB2 and XML file when the Result Set is Huge

Q	DVR	DVX	OrDRR	OrDRX	OpDRR	OpDRX
Q6	1000	1000	2.219	7.25	4.218	-
	15000	1000	4.079	3.375	3.968	-
	15000	15000	1.517	8.743	4.578	-
	1000	15000	0.266	13.453	0.547	-
Q7	1000	1000	0.172	1.609	-	1.062
	15000	1000	0.297	1.218	-	1.172
	15000	15000	0.983	7.797	-	11.719
	1000	15000	6.891	131.328	-	15.672

mal query. But if the access to the XML file is eliminated, the performance is greatly improved. This is due to the characteristics of the relational database. Relational database always has powerful query processing capabilities. By indexing, caching, etc, access and queries to RDB can be finished very quickly. But in the contrast, access to XML files is very slow. There is no inherent optimization technique although nowadays research and industry are devoting themselves into this topic. Discussion about how to optimize XML file access is out of the topic of this work.

3.5.5.4 Predicate Elimination

We design two queries, $Q8$ (Example 3.5.27) and $Q9$ (Example 3.5.28), whose predicates are partially eliminated by our optimizer. $Q8$ is queried over the relational database CSC and $Q9$ is queried over the XML file *xml_customer.xml*. After transformation, Q8 is rewritten into $Q8'$ (Example 3.5.29) and Q9 is rewritten into $Q9'$ (Example 3.5.30).

We use 1000, 5000, and 15000 records respectively in the data sources for testing and get the performance results, as shown in Table 3.13. From Table 3.13, we can see that the predicate elimination technique also provides improvement for the query. The fact that the larger the data volume is, the better the improvement is, still holds.

```
Example Q8:
<results>
{
  for $CSC.CUSTOMER_1 in document("CSO")/db/CUSTOMER_ODDER
  where ($CSC.CUSTOMER_1/TOTAL_ORDER_AMOUNT ge 400)
  return
  <result>
  <CUSTOMER_ID>
    {xf:data($CSC.CUSTOMER_1/CUSTOMER_ID)}
  </CUSTOMER_ID>
  <CITY>
    {xf:data($CSC.CUSTOMER_1/CITY)}
  </CITY>
  </result>
}
</results>
```

Example 3.5.27: Q8

```
Example Q9:
<results>
{
  for $xml_customer.CUSTOMER_1 in
  document("xml_customer_order")/db/CUSTOMER_ORDER
  where ($xml_customer.CUSTOMER_1/TOTAL_ORDER_AMOUNT
   ge 4000)
  return
  <result>
  <CUSTOMER_ID>
    {xf:data($xml_customer.CUSTOMER_1/CUSTOMER_ID)}
  </CUSTOMER_ID>
  <CITY>
    {xf:data($xml_customer.CUSTOMER_1/CITY)}
  </CITY>
</result>
}
</results>
```

Example 3.5.28: Q9

```
Q8'
<results>
{
  for $CSC.CUSTOMER_1 in document("CSC")/db/CUSTOMER
  return
  <result>
  <CUSTOMER_ID>
    {xf:data($CSC.CUSTOMER_1/CUSTOMER_ID)}
  </CUSTOMER_ID>
  <CITY>
    {xf:data($CSC.CUSTOMER_1/CITY)}
  </CITY>
  </result>
}
</results>
```

Example 3.5.29: Q8'

```
Q9':
<results>
{
  for $xml_customer.CUSTOMER_1 in
  document("xml_customer")/db/CUSTOMER
  return
  <result>
  <CUSTOMER_ID>
    {xf:data($xml_customer.CUSTOMER_1/CUSTOMER_ID)}
  </CUSTOMER_ID>
  <CITY>
    {xf:data($xml_customer.CUSTOMER_1/CITY)}
  </CITY>
</result>
}
</results>
```

Example 3.5.30: Q9'

Table 3.13 Performance of Predicate Elimination

Q	DV	OrFT(s)	OpT(s)	RB(%)	Constraint Used
Q8	1000	0.731	0.603	11.35%	IC3
	5000	1.724	1.412	15.49%	
	15000	2.812	2.241	18.71 %	
Q9	1000	1.02	0.812	15.39%	IC6
	5000	4.945	3.701	24.13%	
	15000	15.301	10.885	28.53 %	

3.5.5.5 Discussion

In the reformulation rules for join elimination and predicate elimination, it is possible that the eliminated joins or predicates include attributes which are indexed in the underlying data sources. Exploiting the index typically enhances query performance. If that predicate or join is eliminated, this efficient access might not be chosen anymore. Our optimizer might generate a suboptimal query. Again in the experiments we find that when the query is related only to one underlying relational DBMS and we submit the same query for many times, the query execution becomes always quicker. This is caused by the caching mechanism of RDBMS. So we conclude that when the underlying data source of the query is only one relational DBMS, applying our semantic optimizer might decrease the execution efficiency.

We conclude, that it is probably not worth applying semantic query optimization when the data volume is small and when the query execution cost is expected to be low. However when the data volume is large or when the query execution cost is expected to be high, our optimizer becomes very useful. Another aspect is that in the data integration system, most of the data sources are not RDBMSs, our optimizer works very well. The reason is that the non-relational data sources typically lack indexing mechanism as well as sophisticated processing techniques, therefore, normally the query execution cost is very high. Through the performance tables, we can see that the larger the data volume is, the more benefit can be obtained.

3.6 Summary

Semantic query optimization is an old topic in the database field. In 1990's there was a great deal of work contributing to it. This chapter mainly discussed how to use constraints in the data integration systems to semantically optimize queries submitted to the data integration systems. The first problem we came across was how to express the different constraints coming from heterogeneous data sources. A uniform constraint model was thus established which was based on Active XQuery and which could express all kinds of constraints coming from data sources. In order to translate the different constraints from the data sources into the uniform constraint model, a constraint wrapper was needed. After the translation of the constraint wrapper, the different constraints were stored in the constraint repository. At the same time, if a data source had no constraint system, its administrator could just submit a predicate-like constraint and this constraint would be translated by the constraint wrapper. Again, the administrator of the XML-based data integration system could submit global constraints in the uniform constraint model, too. When a query was submitted to the XML-based data integration system, it would first be transferred to the query optimizer. The architecture of the query optimizer was given. The query adapter, which was the most important component in the optimizer, was discussed in detail. It was composed of query decomposer, constraints fetcher, conflict detector and query reformulator. Three semantic query optimization techniques were implemented in the query reformualtor: detection of empty results, predicate elimination and join elimination. Experiments were carried on and the performance results were given in tables. In general, performance of the XML-based data integration system was greatly improved by the optimizer when the data volume was huge and when the data sources were not relational.

Chapter 4

Updates in XML-DIS

In order to explain why data consistency enforcement is needed in a data integration system, we must first go to the updates in the XML-based data integration. We will begin with updating the views in both relational databases and XML-based data integration systems. Then we will present the W3C XQuery Update Facilities. The Service Data Objects programming framework will be proposed at the end.

4.1 Updating the View

4.1.1 Updating Relational Views

Most relational database systems permit users to delimit their view of the database to those portions that are relevant to their applications. Views enhance logical data independence, since most changes in the structure of the database need not impact a view and hence, do not impact the application programs that use the view. Views also simplify the user interface by allowing the user to ignore data which they have no interest in. Further, views provide a measure of protection by preventing the user from accessing data outside the view.

The structure of the database is defined by the *conceptual schema*. The contents of the database (e.g., sets of tuples) are called the *extension of the conceptual schema*. The structure of a view is defined by a sequence of operations applied to the conceptual schema. The contents of the view, called the *extension of the view*, are defined by the same sequence of operations applied to the *conceptual schema extension*. This sequence of operations is a functional mapping from *conceptual schema* extensions to view extensions and is called the *view definition function* [DB82].

A view extension does not have an independent existence. It is the conceptual schema extension that actually exists in the database. The view extension is completely defined by applying the view definition function to the current conceptual schema extension. Because this mapping is functional, updates on the conceptual schema extension are translated unambiguously into corresponding changes in the view extension.

For a view to be useful, users must be able to apply retrieval and update operations to it. These operations on the view extension must be translated into corresponding operations on the conceptual schema extension. Retrievals from a view extension can always be mapped into equivalent retrievals from the conceptual schema extension: in order to evaluate a retrieval query against a view in an abstract (and not optimized) model, one must first construct the view extension by applying the view definition function to the conceptual schema extension, and then evaluate the query against the view extension. Clearly, this procedure retrieves exactly the data requested by the query.

A mapping is also required to translate view updates into corresponding updates on the conceptual schema extension. However, such an update mapping does not always exist, and even when it does exist, it may not be unique. So a change in the view extension may not be reflected unambiguously by equivalent changes in the database. There are five approaches to the problem of mapping relational view updates:

1. View as an abstract data type: In this approach the DBA defines the view together with the updates it supports. The effect of updates on the base relations is explicitly defined [RS86] [TFC83]. The view definition describes not only how view data are derived from the conceptual schema extension, but also how operations on the view are mapped into operations on the conceptual schema. This approach requires a method for designing views and their operational mappings and for verifying that the design is correct,

which means, the conceptual schema operations indeed perform the desired view operations correctly.

2. To define general translation procedures. These procedures take as input a view definition, a view update, and the current schema extension and produce, if possible, a translation of the view update into conceptual schema updates satisfying some desired properties. [DB82] proposes a translation mechanism that uses view graphs to decide if a given update translation is correct. The view graphs are constructed based on the syntax of the view definition and on the functional dependencies of base relations.
3. View complement. [BS81] introduces the notion of view complement to solve the update problem. Together with the user-defined view, a "complementary" view is defined such that the database could be computed from the view and its complement. A view can have many different complements and the choice of a complement determines an update policy. An update translation is considered correct if the complement of the view remains constant. Finding a view complement may be NP-complete even for very simple view definitions [CP84].
4. View as conditional tables. In [Shu00] views are represented as conditional tables. It transforms a view update into a constraint satisfaction problem. Each solution to the constraint satisfaction problem corresponds to a possible translation of the view update.
5. Object-based views. An extension of [Kel86] to deal with object-based views is proposed in [BSKW91]. Algorithms are proposed for propagating updates in a hierarchical structure of objects.

4.1.2 Updating XML Views of Relational Data

XML is frequently used for publishing as well as exchanging data. Due to the high unintuitive representation of data in the relational model, it is also increasingly being used as a mechanism through which to query and update legacy relational databases.

The problem of updating XML views published over relational data comes with new challenges beyond those of updating relational views. The challenges relate to the translatability problem. That is, the mismatch between the hierarchical XML view model and the flat relational base model raises the question whether the given view update is even mappable into SQL updates. For instance, the nested structure imposed by an XML view may be in conflict with the constraints of the underlying relational schema, so that updates that are valid on the view may not be achieved on the base. The second challenge concerns the translation strategy issue. That is, assuming the view update is indeed identified as being translatable, one needs to devise a strategy to identify the minimal mapping update. This mapping has to best bridge the two query and update languages (updates with diverse granularity on the XML view versus flat tuple-based SQL updates on the relational base).

The update translatability question remains largely unexplored. [WRM06] addresses the question, if for a given update over an XML view, a correct relational update translation exists. Intuitively, the translatability of XML view update question can be described as follows: given a relational database and an XML view definition over it, is it possible to decide whether an update against the XML view is translatable into corresponding updates against the underlying relational database without violating any consistency? Intuitively, consistency means that (i) the requested view update is valid. That is, it agrees with the implied XML view schema. (ii) The translated updates against the relational database comply with the relational schema, namely, to keep the relational database consistent by update propagation if necessary. (iii) The XML view reconstructed on the updated relational database using the view definition is exactly the same as the result that would be generated by directly updating the materialized view, namely, without view side-effects.

Commercial database systems, such as Oracle, DB2 and SQL-Server, also provide XML support. Oracle XML DB [Ora00] provides SQL/XML as an extension to SQL, using functions and operators to query and access XML content as part of normal SQL operations, and also to provide methods for generating XML from the result of an SQL Select statement. The IBM DB2 XML Extender [CX01] provides user-defined functions to store and retrieve XML documents in XML columns, as well as to extract XML elements or attribute values. However, neither IBM nor Oracle support update operations. [Rys01] introduces XML view updates in SQL Server2000, based on a specific annotated schema and update language called updategrams, under the assumption of updates always being translatable. Annotated schemata consist of a schema description of the exposed XML view and annotations that describe the mapping of the XML schema constructs onto the relational schema constructs. Instead of using update statements, SQL Server 2000 uses updategrams, which include a before and after image of the view to compute the corresponding SQL statements. Updategrams provide an intuitive way to perform an instance-based transformation from a before state to an after state. Updategrams operate over either a default XML view implied by its instance data (if no annotated schema is referenced) or over the view defined by the annotated schema and the top-level element of the updategram. Example 4.1.1 is a simple example.

```
<root xmlns:updg=''urn:schema-microsoft-com:xml-updategram''>
 <updg:sync mapping-schema=''nwind.xml'' nullvalue=''ISNULL''>
      <updg:before>
        <Customer CustomerID=''LAZYK''
         CompanyName=''ISNULL''
         Address=''Reinsburgstrasse 134a''>
        <Order OID=''10482''/>
        </Customer>
      </updg:before>
     <updg:after>
        <Customer CustomerID=''LAZYK''
         CompanyName=''UniCompany''
         Address=''Universitaetsstr.38''>
        <Order OID=''12345''/>
        </Customer>
      </updg:after>
    </updg:sync>
  </root>
```

Example 4.1.1: Updategram in SQL Server 2000

Updategrams use their own namespace *urn:schema-microsoft-com:xml-updategram*. Each *updg:sync* block defines the boundaries of an update batch that uses optimistic concurrency control [Mic09] to perform the updates transactionally. The before image in *updg:before* is used both for determining the data to be updated as well as to perform the conflict test. The after image in *updg:after* provides the state to which the data has to be changed. If the before state is empty or missing, the after state defines an insert; if the after state is empty or missing, the before state defines what should be deleted. Otherwise the necessary insertions, updates and deletions are inferred from the difference between the before and after image.

In Example 4.1.1, the customer with the given data (including a company name set to NULL) gets a new company name and address. In addition, the relation to the order 10482 is removed and replaced by a new relational to order 12345.

[BDH04] studies the XML view update problem using a nested relational algebra. They assume the view is always well-nested, that is, joins are through keys and foreign keys, and nesting is controlled to agree with the integrity constraints and to avoid duplication. The update over such a view is thus always translatable. Unfortunately, in general, conflicts are possible and a view cannot always be guaranteed to be well-nested. Our work is under the assumption that the update on the XML view is translatable.

4.1.3 Updating XML Views in XML-based Data Integration Systems

Because in an XML-based data integration system, there is no materialized data stored in it, realizing update in the XML-based data integration system is to realize update in a virtual view. Figure 4.1 shows an XML-based data integration system which supports updates.

Realizing update in XML-based data integration system includes two situations: one is to support XQuery updates in the XML-based data integration system, the other is to support normal updates from the end applications (e.g., Java applications, web services, etc). If the users submit XQuery updates, the data integration system is responsible to parse the XQuery updates (an XQuery Update Engine is then needed), divide them into sub-updates and transform the sub-updates to the wrappers of each corresponding data sources. The wrappers then translate the sub-XQuery-Updates into the local update language or interact with the local update interfaces to finish the local updates. Notice that when the users submit updates, there will be no results

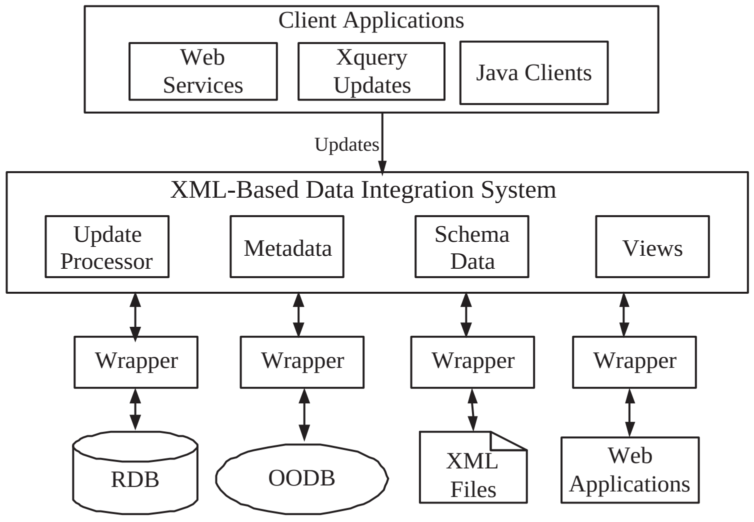


Fig. 4.1 Updates in an XML-based Data Integration System

returned. Some message mechanism might be needed to let users know the execution results: succeed or fail.

If the users submit updates from Java Applications or Web services, the updates need also to be analyzed, divided and transformed to the wrappers. We will introduce XQuery Updates and realizing updates by Service Data Objects in the following sections.

4.2 XQuery Update Facility 1.0 by W3C

Over the past several years, the growth of the Web and e-commerce has stimulated a tremendous surge of interest in XML as a universal, queryable representation for data. Now XML is almost the *de facto* standard for information interchange. Nearly every vendor of data management tools has added support for exporting, viewing and in some cases even importing, XML-formatted data. XML document repositories like Software AG's Tamino [AG09] and MarkLogic [Mar08c] are now available, and XML publishing capabilities have been added to the relational database systems from Oracle (Oracle Business Intelligence Publisher (BI Publisher, formerly XML Publisher)) [Ora08a], IBM [Mel05] and Microsoft [SPD08].

Ultimately, it is expected that these relational database engines will provide standardized integration support for querying and publishing XML views of databases. This will allow the sharing of data from both XML repositories and traditional relational databases using a single, unified queryable model - namely, XML views with an XML query language. The next step in making XML to a full-featured data exchange format is to support not only queries, but updates, over XML content. It should be possible to modify content within XML documents and to express updates to XML views. The ability to encapsulate an update operation is also necessary for expressing incremental changes ("deltas") over content, which is important for Continuous Queries [CDTW00], XML document mirroring, caching, and replication.

Therefore, by W3C, XQuery Update Facility 1.0 defines an update facility that extends the XML Query language, XQuery. The XQuery Update Facility provides expressions that can be used to make persistent changes to instances of the XQuery 1.0 and XPath 2.0 Data Model.

We give a short introduction of XQuery Update syntax and some explaining examples below.

4.2.1 INSERT

The Syntax of XQuery Update INSERT is shown in Syntax 1.

```
InsertExpr::="insert" ("node" | "nodes")
               SourceExpr InsertExprTargetChoice TargetExpr
InsertExprTargetChoice::=(("as" ("first" | "last"))? "into")
                         | "after"
                         | "before"
SourceExpr     ::=     ExprSingle
TargetExpr     ::=     ExprSingle
```

Syntax 1: XQuery Update INSERT

The semantics of INSERT can be found in [W3C08a]. We give no explanation of the semantics here, but give an example to show how to use the INSERT of XQuery Update Facility.

Example 4.2.1: Insert a year element after the publisher of the first book in a given bibliography (bib.xml).

```
    insert node <year>2005</year>
        after fn:doc("bib.xml")/books/book[1]/publisher
```

Example 4.2.1: XQuery INSERT

4.2.2 DELETE

The Syntax of XQuery Update DELETE is shown in Syntax 2.

```
     DeleteExpr::="delete" ("node" | "nodes") TargetExpr
     TargetExpr::=ExprSingle
```

Syntax 2: XQuery Update DELETE

Example 4.2.2: Delete the last author of the first book in a given bibliography.

```
    delete node fn:doc("bib.xml")/books/book[1]/author[last()]
```

Example 4.2.2: XQuery DELETE

4.2.3 REPLACE

The Syntax of XQuery Update REPLACE is shown in Syntax 3.

```
     ReplaceExpr::= "replace" ("value" "of")? "node" TargetExpr
                    "with" ExprSingle
     TargetExpr ::= ExprSingle
```

Syntax 3: XQuery Update REPLACE

4.2.3.1 Replacing a Node

Example 4.2.3: Replace the publisher of the first book with the publisher of the second book.

```
replace node fn:doc("bib.xml")/books/book[1]/publisher
with fn:doc("bib.xml")/books/book[2]/publisher
```

Example 4.2.3: XQuery Update REPLACE (1)

4.2.3.2 Replacing the Value of a Node

Example 4.2.4: Increase the price of the first book by ten percent.

```
replace value of node fn:doc("bib.xml")/books/book[1]/price
with fn:doc("bib.xml")/books/book[1]/price * 1.1
```

Example 4.2.4: XQuery Update REPLACE (2)

4.2.4 RENAME

The Syntax of XQuery Update RENAME is shown in Syntax 4.

```
RenameExpr::="rename" "node" TargetExpr "as" NewNameExpr
TargetExpr::=ExprSingle
NewNameExpr::=ExprSingle
```

Syntax 4: XQuery Update RENAME

Example 4.2.5: Rename the first author element of the first book to principal-author.

```
rename node fn:doc("bib.xml")/books/book[1]/author[1]
as "principal-author"
```

Example 4.2.5: XQuery Update RENAME

4.2.5 State-of-art of XQuery Update

On March 28, 2008, W3C has published the XQuery Update Facility as a candidate recommendation. Unfortunately, until now the W3C has not yet published the proposed recommendation of XQuery Update Facility standards.

The consequence is, most of the XML repository vendors and XML-based data integration systems do not support accepting XQuery Update Facility now recommended by W3C. In XML repository products, e.g., Software AG's Tamino XML Server supports it own XQuery Update syntax; MarkLogic only supports its own built-in update functions in the libraries [Mar08b]. In XML-based data integration systems, e.g., BEA AquaLogic DataService Platform, none supports XQuery Updates directly. Generally, XQuery Update Facility is still in its infancy. Next, we will introduce how to realize update in an XML-based data integration system through a programming framework: Service Data Objects.

4.3 Realizing Updates through SDO

4.3.1 SDO Goals

Service Data Objects (SDO) is a specification for a programming model that unifies data programming across data source types, provides robust support for common application patterns, and enables applications, tools, and frameworks to more easily query, view, bind, update, and introspect data.

While the Java platform and J2EE provide a variety of data programming models and APIs, these technologies are fragmented and are not always amenable to tooling and frameworks. Further, some of the technologies can be hard to use and may not be sufficiently rich in functionality to support common application needs. SDO is intended to create a uniform data access layer that provides a data access solution for heterogeneous data sources in an easy-to-use manner that is amenable to tooling and frameworks. SDO is not motivated to replace lower-level data access technologies.

The goals of SDO are as following:

- Unified data access to heterogeneous data sources. Current data programming technologies are more or less specific to data source types. In real-world applications, especially in a data integration system, however, data frequently comes from a variety of sources.

- Unified support for both static and dynamic data APIs. Static, strongly typed interfaces provide an easy-to-use programming model for application programmers. ResultSet and RowSet interfaces in JDBC, in contrast, provide only dynamic, untyped data APIs. Neither static nor dynamic data APIs alone are sufficient, however, both are necessary.
- Support for tools and frameworks. Current data programming technologies are not amenable to tools and frameworks across data source types.
- Support for disconnected programming models. Many applications naturally have a disconnected usage pattern of data access: an application reads a set of data, retains it locally for a short period of time, manipulates the data, and then applies the changes back to the data source.
- Support for custom data access layers based on common design patterns. Many applications use well-known design patterns (e.g., Transfer Object [Sun08c], Transfer Object Assembler [Sun08d], and Data Access Objects [Sun08a]) to build custom data access layers. These patterns are commonly used when the application needs to be insulated from the physical data sources. Implementing data access layers commonly requires a significant amount of custom code, much of which can be automated. Further, these custom data access layers should be able to be integrated into tools and frameworks.
- Decouple application code from data access code. In order to be reusable and maintainable, application code should be separable from data access. Data access technologies should be amenable to this separation of concerns.

4.3.2 SDO Preliminaries

The SDO architecture is based upon the concept of disconnected data graphs. Under the disconnected data graphs pattern, a client retrieves a data graph from a data source, mutates the data graph, and can then apply the data graph changes back to the data source. Most commonly, the update is performed with optimistic concurrency semantics [Mic09], which means that if any of the underlying data was changed before the client applies the changes. the update is rejected and the application must take corrective action. Optimistic concurrency semantics is a natural semantic that serves most business applications well.

The key concepts in the SDO architecture are the data object, the data graph and the Data Access Services (DAS) [BIO$^+$], as shown in Figure 4.2. Additionally, metadata plays a key role.

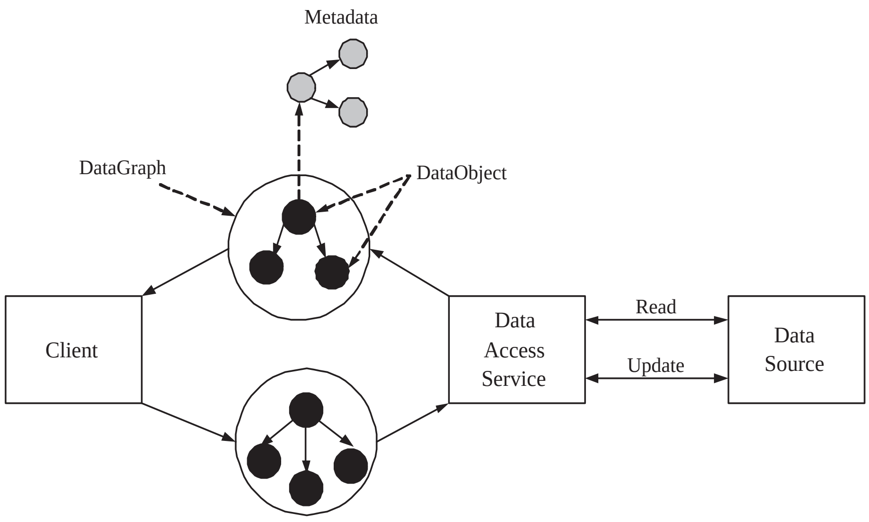


Fig. 4.2 Components of the SDO Solution

These components have the following responsibilities:

- Data Objects. A data object holds a set of named properties, each of which contains either a simple data-type value or a reference to another data object. The data object API provides a dynamic data API for manipulating these properties.
- Data Graph. The data graph provides an envelope for data objects. It is the normal unit of transport between components. Data graphs can track changes made to the graph of data objects. Changes include inserting data objects, deleting data objects and modifying data object property values. Usually, data graphs are constructed from one of the following:

  1. Data sources such as XML files, Enterprise Java Beans (EJBs), XML databases and relational databases.
  2. Services such as Web services, Java Connector Architecture (JCA) Resource Adapters [Sun08b] and Java Message Service (JMS) messages [Sun09].

- Metadata. Metadata about data objects enables development tools and runtime frameworks to introspect data, including data types, relationships, and constraints. SDO provides a common metadata API across data source types to enable generic tools and frameworks.
- Data Access Service. Components that can populate data graphs from data sources and commit changes to data graphs back to the data source are called Data Access Services (DAS).

4.3.3 Data Graph

Data graphs provide a container for a tree of data objects. They are produced by the DAS for SDO clients to work with. Once modified, data graphs are passed back to the DAS for updating the data source. SDO clients can traverse a data graph and read and modify its data objects. SDO is a disconnected architecture, because SDO clients are disconnected from the DAS and the data source. They only see the data graph. Furthermore, a data graph can include objects representing data from different data sources. A data graph contains a root data object, all of the root's associated data objects, and a change summary. When being transmitted between application components (for example, between a Web service requester and provider during service invocation), to the DAS, or saved to disk, data graphs are serialized to XML. The SDO specification provides the XML Schema of this serialization (see **Appendix A**). Figure 4.3 shows an SDO data graph.

Change summaries are part of data graphs and are used to represent the changes that have been made to a data graph returned by the DAS. They are initially empty (when the data graph is handed over to a client) and populated as the data graph is modified. Change summaries are used by the DAS at backend update time to apply the changes back to the data source. They allow DASs to efficiently and incrementally update data sources by providing lists of the changed properties (along with their old values) and the created and deleted data objects in the data graph.

A data graph may be serialized as an XML stream. In general, the data graph serialization consists of a description of the schema used for the data

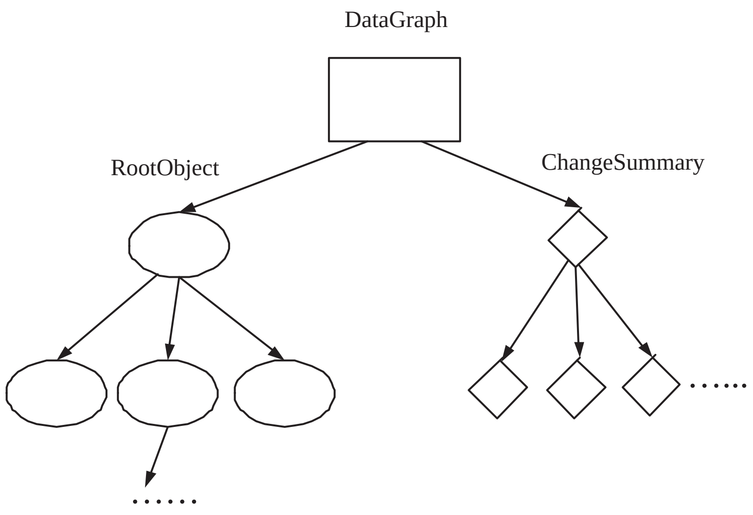


Fig. 4.3 An SDO Data Graph

graph, followed by the data objects that are contained in the data graph, followed by a description of the changes. The serialization of data objects follows the XMI specification [Gro07] or the XSD for the data object model, producing the same XML stream independent of the enclosing data graph element. When XML Schema is used as the metadata, the XML serialization of the data objects follow the XSD and the resulting XML elements should validate with the XML Schema when all the constraints for the XSD are enforced.

The description of the schema is optional and can be expressed either as an XSD or EMOF model [OMG06]. We use XSD for explanation here. The description of the changes is also optional. The changes are expressed as a change summary. XSDs are typically included if it is likely that the reader of the data graph would not be able to retrieve the model by the logical URI of the XSD targetNamespace. The optional serialization of the ChangeSummary follows XMI, where properties that have not changed values are omitted. When serializing XSDs, only the XSDs actually used by the data objects are typically transferred. When the data graph was originally created from an XSD, the XSD form is preferred in order to preserve all original XSD information. If the data graph is from a source other than XSD, an XSD may be generated (typically following the EMOF and XMI specifications) and included, or the EMOF model may be included. The choice of which to include is determined by the serializer of the data graph.

4.3.4 Change Summary

The serialization of the ChangeSummary includes enough information to reconstruct the original information of the data objects at the point when logging was turned on. The goal of this format is to provide a simple XML representation that can express the difference between the graph when logging begins and ends.

4.3.4.1 Creating Data Objects

The ChangeSummary which creates data objects is normally with the head shown in Figure 4.4:

```
<changeSummary create="   TargetExpression  "/>

   TargetExpress = A set of XPath expressions
```

Fig. 4.4 Head of Change Summary: Creating Data Objects. (INSERT)

"Create" in the ChangeSummary means to create data objects. The "TargetExpression" is a set of XPath expressions. Each XPath expression specifies the location path of a created data object. In case of creating data object, there are no old values that are recorded in the ChangeSummary. The newly created Root data object is a sibling node of ChangeSummary element stored in the data graph. Figure 4.5 shows an example where five data objects are created, which are the Root data object "Profile" and its four descendant data objects including a "Student" data object, two "Address" data objects, and a "Postcode" data object. The ChangeSummary tracks every location path of the created Root data object and its descendants in the attribute value. The number used in the location path specifies the sequence of the child nodes in the tree.

```
<!-- ChangeSummary -->
<changeSummary create="/Profile
                  /Profile/Student[1]
                  /Profile/Student[1]/Address[1]
                  /Profile/Student[1]/Address[2]
                  /Profile/Student[1]/Address[2]/PostCode"/>
<!-- Root Data Object -->
<ns0:Profile xmlns:ns0="ld:DataServices
   /StudentProfileXML/StudentProfile">
      <Student>
         <StudentID>09094201</StudentID>
         <Name>Jing Lu</Name>
         <Address>
            <HomeStreet>Reinsburgstr</HomeStreet>
            ...
         </Address>
         <Address>
            <OfficeStreet>Universitaetsstr</OfficeStreet>
            <PostCode>70569</PostCode>
            ...
         </Address>
      </Student>
</ns0:Profile>
```

Fig. 4.5 Example of Change Summary: Creating Data Objects. (INSERT)

4.3.4.2 Deleting Data Objects

The ChangeSummary recording deletion operation has the head shown in Figure 4.6.

The ChangeSummary element uses an attribute “delete” to record the deletion operation in the data graph. An automatically generated identifier for the deleted data object is assigned to the value of this attribute. In case of deleting data objects, the ChangeSummary element contains a copy of the deleted data objects, which refers to the old values generated by the deletion operation. Figure 4.7 is an example of ChangeSummary which deletes data objects. “log.0” is the automatically generated identifier that identifies the

```
<changeSummary delete="#D">
   <A com:id="D">
      ...
   </A>
   ...
</changeSummary>

A = Name of deleted Data Object
D = A set of identifiers for the deleted Data Objects
```

Fig. 4.6 Change Summary: Deleting Data Objects. (DELETE)

to-be-deleted "Employee" data object in the ChangeSummary. All properties and their values of this deleted data object are recorded as old values in the ChangeSummary.

```
<changeSummary delete="#log.0">
   <emp:Employee com:id="log.0"
            xmlns:emp="ld:DataServices
                   /EmployeeDB/Employee">
      <EmpID>00010</EmpID>
      <Name>John Smith</Name>
      <WorkDeptID>A00</WorkDeptID>
      <Salary>52750</Salary>
   </emp:Employee>
</changeSummary>
```

Fig. 4.7 Example of Change Summary: Deleting Data Objects. (DELETE)

The "delete" attribute can also record multiple deleted data objects. This situation can occur only when underlying data source has a complex multi-layer structure. For example, the "delete" attribute records a set of identifiers of deleted data object, when a set of child nodes in an XML document are deleted. Recording multiple identifiers of deleted data objects in the Change-Summary is shown in Figure 4.8.

```
<changeSummary delete="#log.0 #log.1">
  <!-- updated father Data Object -->
  <Profile com:ref="/Profile">
    <Customer com:id="log.0">
      <!-- deleted child Data Object -->
        <CustomerID>C3456</CustomerID>
        ...
    </Customer>
    <Customer com:id="log.1">
          <!-- deleted child Data Object -->
        <CustomerID>C3457</CustomerID>
        ...
    </Customer>
  </Profile>
</changeSummary>
```

Fig. 4.8 Change Summary: Deleting Two Data Objects Example. (DELETE)

4.3.4.3 Updating Data Object Property Values

The ChangeSummary which updates data objects has the head shown in Figure 4.9.

```
 <changeSummary>
   <A com:ref="C">
       <B>{old property value}</B>
   <A/>
   ......
 </changeSummary>
<!-- Root Data Object-->
New property values

 A = Name of updated Data Object
 B = Property name
 C = Target XPath expression
```

Fig. 4.9 Head of Change Summary: Updating Data Objects. (UPDATE)

When updating data objects, ChangeSummary must record both the old values and the new values. “A” is the name of the updated data object. “B” records the property name and “C” is the target XPath expression. Then the new values are recorded in the root data objects. Figure 4.10 is an explaining example, which shows how to update an XML file which stores all customer information and has a multi-layer structure. In the ChangeSummary it can be seen that four data objects are updated: one “Customer” data object, two “Address” data objects, and one “Street” data object. Every data object has a changed property. These four properties are: the name of the first customer, the zip of the first address, the city of the second address, and the street name of the second address. The old values of these four properties are recorded in the ChangeSummary element. The new values can be found in the Root data object. The ChangeSummary tracks the location path of each updated data object in its attribute value. The number used in the location path indicates that the location path uses abbreviated syntax to specify the sequence of child nodes in the tree.

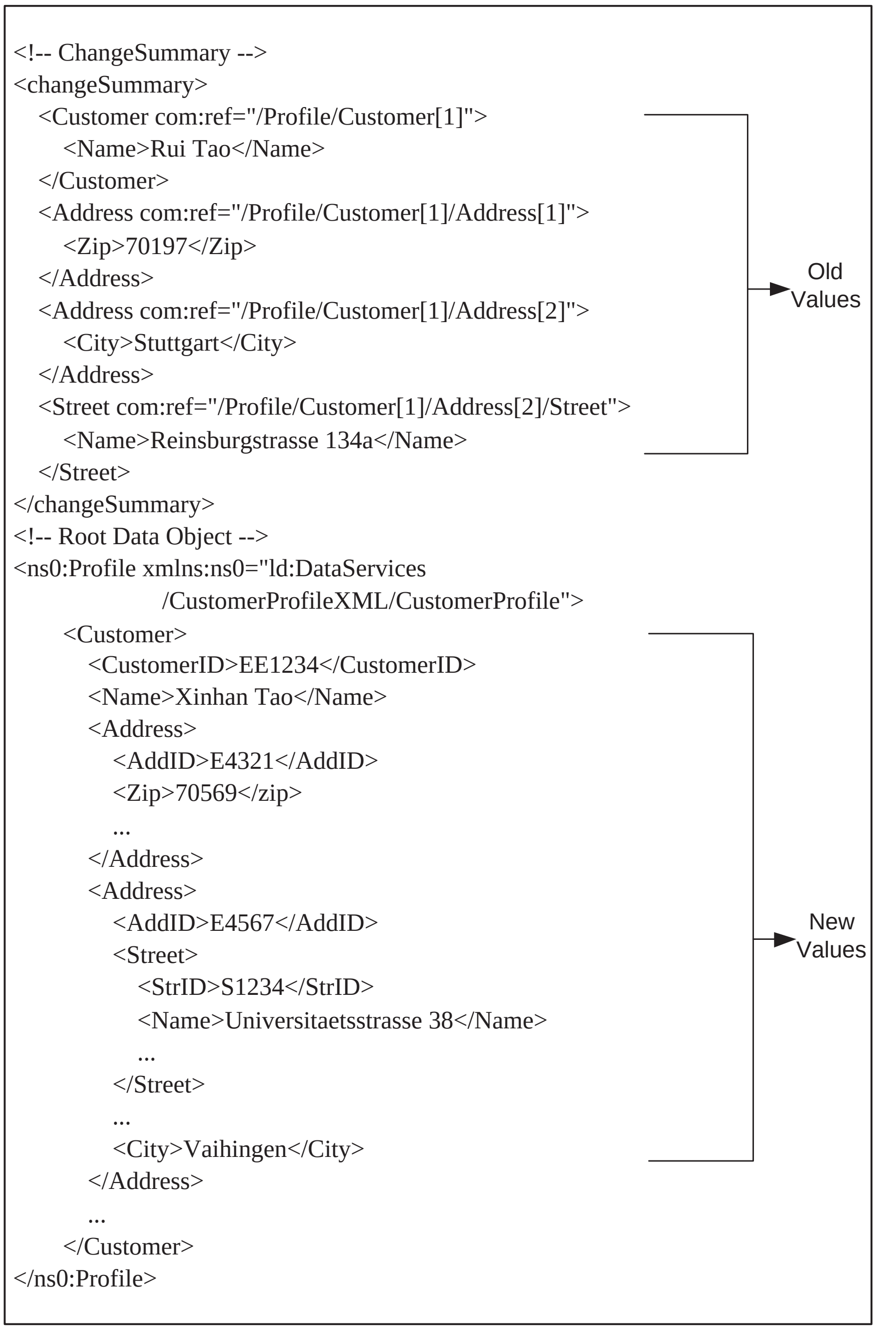
4.3.4.4 Hybrid Modifications

Hybrid modifications consisting of INSERT, DELETE, and UPDATE are also supported. These modifications can be applied to the data object and all its descendant data objects. The ChangeSummary records the hybrid modifications applied to the data object and all old values need to be preserved. The new values are recorded in the Root data object. Figure 4.11 illustrates a ChangeSummary instance that is generated after applying hybrid modifications to a “Customer” data object. The explanation is listed, too.

4.4 Service Data Objects in XML-based Data Integration Systems

4.4.1 DAS in DIS

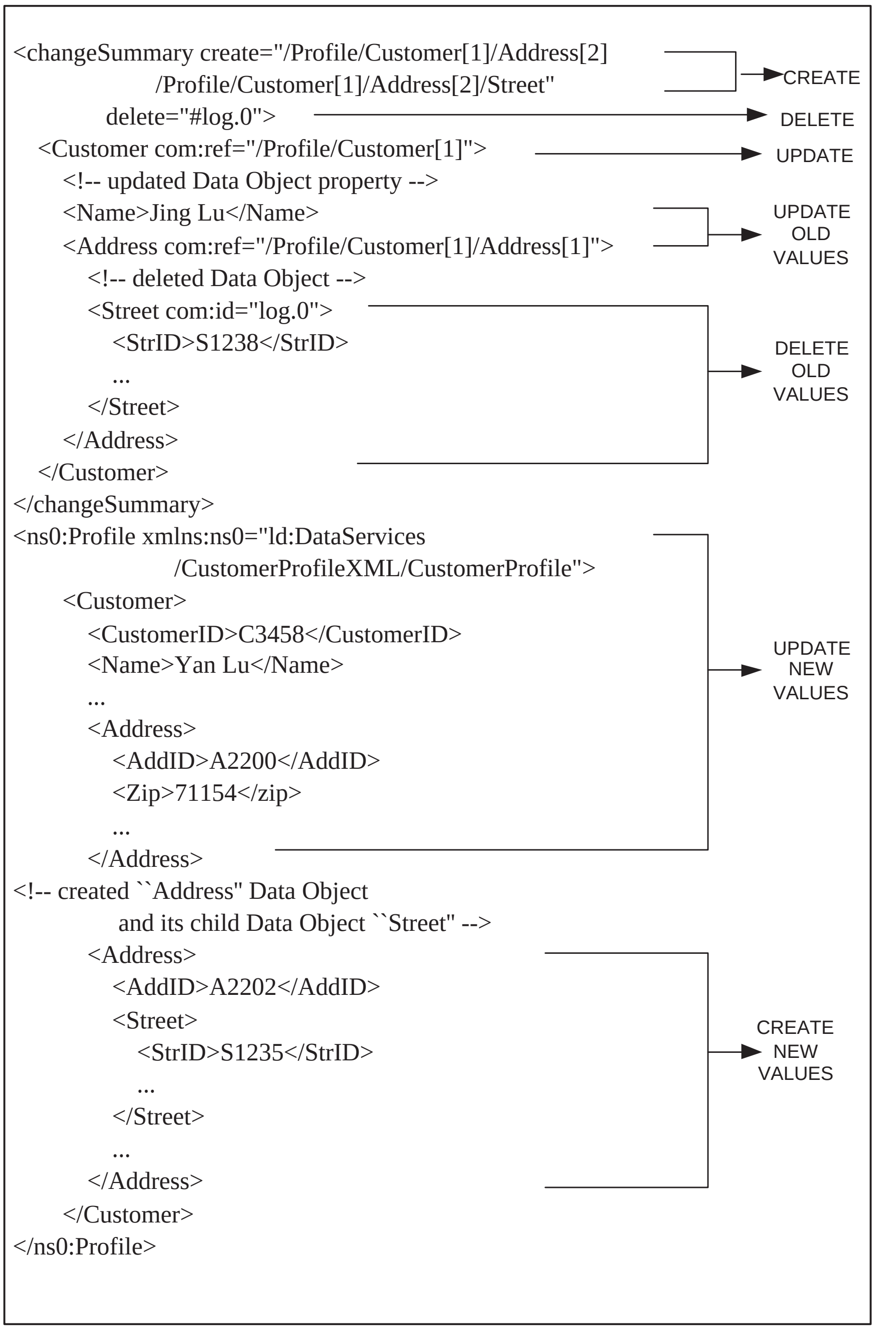
Refer to Figure 4.1 again, in order to support retrieval and especially update in the XML-based data integration system, a Data Access Service must be running in the XML-based data integration system. Figure 4.12 shows an



```
<!-- ChangeSummary -->
<changeSummary>
  <Customer com:ref="/Profile/Customer[1]">
    <Name>Rui Tao</Name>
  </Customer>
  <Address com:ref="/Profile/Customer[1]/Address[1]">
    <Zip>70197</Zip>
  </Address>
  <Address com:ref="/Profile/Customer[1]/Address[2]">
    <City>Stuttgart</City>
  </Address>
  <Street com:ref="/Profile/Customer[1]/Address[2]/Street">
    <Name>Reinsburgstrasse 134a</Name>
  </Street>
</changeSummary>
<!-- Root Data Object -->
<ns0:Profile xmlns:ns0="ld:DataServices
            /CustomerProfileXML/CustomerProfile">
    <Customer>
      <CustomerID>EE1234</CustomerID>
      <Name>Xinhan Tao</Name>
      <Address>
        <AddID>E4321</AddID>
        <Zip>70569</zip>
        ...
      </Address>
      <Address>
        <AddID>E4567</AddID>
        <Street>
          <StrID>S1234</StrID>
          <Name>Universitaetsstrasse 38</Name>
          ...
        </Street>
        ...
        <City>Vaihingen</City>
      </Address>
      ...
    </Customer>
</ns0:Profile>
```

Old Values
New Values

Fig. 4.10 Example of Change Summary: Updating Data Objects. (UPDATE)



```
<changeSummary create="/Profile/Customer[1]/Address[2]
                  /Profile/Customer[1]/Address[2]/Street"
            delete="#log.0">
   <Customer com:ref="/Profile/Customer[1]">
      <!-- updated Data Object property -->
      <Name>Jing Lu</Name>
      <Address com:ref="/Profile/Customer[1]/Address[1]">
         <!-- deleted Data Object -->
         <Street com:id="log.0">
            <StrID>S1238</StrID>
            ...
         </Street>
      </Address>
   </Customer>
</changeSummary>
<ns0:Profile xmlns:ns0="ld:DataServices
                    /CustomerProfileXML/CustomerProfile">
      <Customer>
         <CustomerID>C3458</CustomerID>
         <Name>Yan Lu</Name>
         ...
         <Address>
            <AddID>A2200</AddID>
            <Zip>71154</zip>
            ...
         </Address>
<!-- created ``Address" Data Object
             and its child Data Object ``Street" -->
         <Address>
            <AddID>A2202</AddID>
            <Street>
               <StrID>S1235</StrID>
               ...
            </Street>
            ...
         </Address>
      </Customer>
</ns0:Profile>
```

Fig. 4.11 Change Summary: Hybrid Example.

XML-based data integration system where SDO is used as the programming framework.

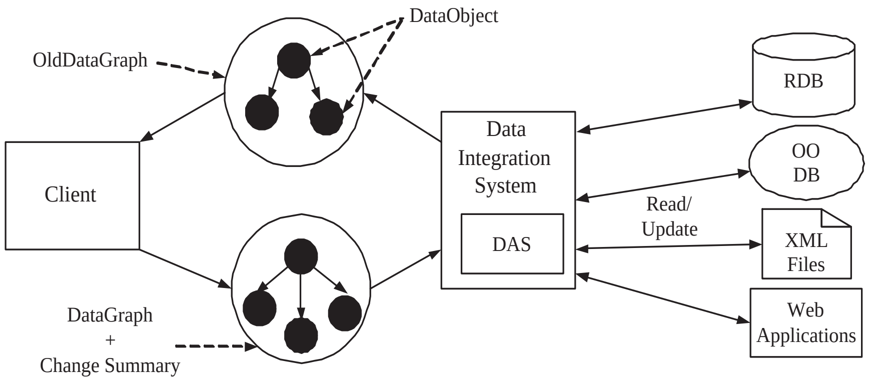
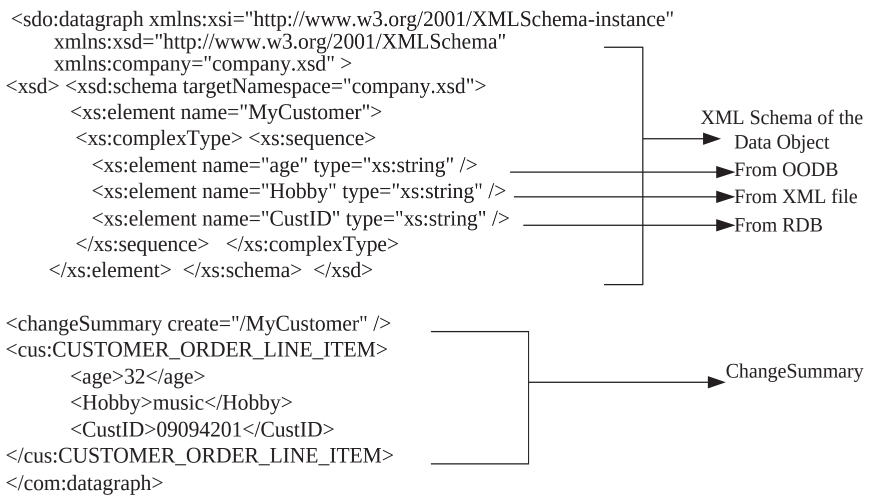


Fig. 4.12 An XML-based Data Integration System Supporting SDO

4.4.2 Global Update in DIS by SDO

Figure 4.13 shows a data graph which acts as a view on three data sources: OODB, XML File and RDB, as shown in Figure 4.14. This data graph is serialized in XML. The global view *myCustomer* is composed of three elements: *age* comes from OODB, *Hobby* comes from XML file and *CustID* comes from RDB.



```
 <sdo:datagraph xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xmlns:xsd="http://www.w3.org/2001/XMLSchema"
        xmlns:company="company.xsd" >
<xsd> <xsd:schema targetNamespace="company.xsd">
          <xs:element name="MyCustomer">
           <xs:complexType> <xs:sequence>
             <xs:element name="age" type="xs:string" />
             <xs:element name="Hobby" type="xs:string" />
             <xs:element name="CustID" type="xs:string" />
           </xs:sequence>   </xs:complexType>
      </xs:element>  </xs:schema>  </xsd>

<changeSummary create="/MyCustomer" />
<cus:CUSTOMER_ORDER_LINE_ITEM>
          <age>32</age>
          <Hobby>music</Hobby>
          <CustID>09094201</CustID>
</cus:CUSTOMER_ORDER_LINE_ITEM>
</com:datagraph>
```

XML Schema of the Data Object
From OODB
From XML file
From RDB
ChangeSummary

Fig. 4.13 Data Graph as an Integration View

When the data integration system supports SDO, a Data Access Service will run in it. Figure 4.14 shows how an update is submitted, analyzed, decomposed and executed in the data integration system where a Data Access Service is running. From Figure 4.14, we can see that the client will submit a data graph including three elements: Age, Hobby and CustID. The Data Access Service will analyze the changes in the data graph and will carry on the decomposition. After the processing of the Data Access Service, *age* goes to OODB, *Hobby* goes to XML file and *CustID* goes to RDB. The changes will be executed in the local data sources.

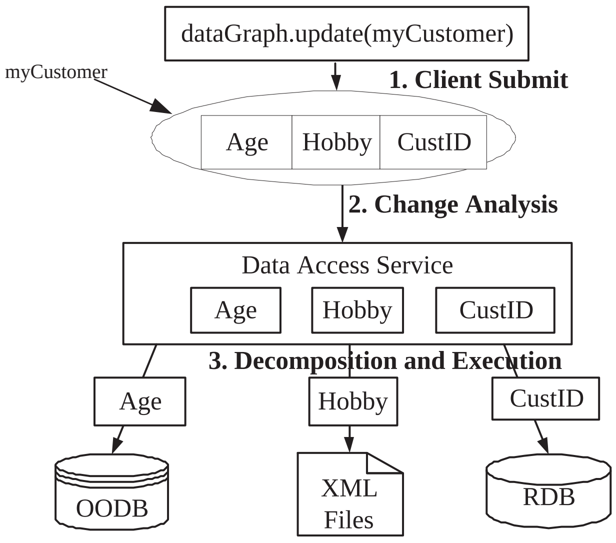


Fig. 4.14 Update on the Integration View by SDO

4.5 Data Services: Combine SDO and SOA

Service-Oriented Architecture (SOA) [IBM08] is considered as a loosely coupled but not completely decoupled architecture, because, while the service provider has no knowledge of the interfaces or business concerns of its consumers, the consumers do include explicit references to the service provider interface, in the form of service calls. The benefits of loose coupling include simplified data access, reusability, and adaptability. New services can be exposed and used without requiring extensive changes to existing applications. The result is a service layer that is highly adaptive and change tolerant.

Data Service is now accepted as a popular idea to give consumers an easy-to-use and uniform model for accessing heterogeneous, distributed data. BEA [BEA08a] and DataDirect [Dat08] proposed this idea and now data services are becoming more and more popular in accessing heterogeneous data. Data services fit perfectly into the SOA picture by providing a data abstraction layer between data users and the underlying data sources. Thus the application developer needs to work only with a uniform, well-defined SOA-based data access interface. The capabilities of data services are the following:

1. Insulate integrated applications and processes from complexity of divergent data forms and disconnected heterogeneous sources of enterprise data.
2. Manage the metadata information imported from disparate data sources.
3. Create data models showing the relationships between various data services.

There are two types of data services: physical data services and logical data services. The underlying are the heterogeneous and disconnected data sources. The physical data services are built directly above the data sources. The logical data services are build upon one or more physical data services. The logical data services are also built upon logical data services or as a combination of both physical and logical data services. The data access layer of the enterprise includes both logical and physical data services.

A data service is similar to a conventional Web service in the following respects:

1. It consists of public functions.
2. The functions that access services are modular, reusable, and extensible.
3. Implementation details are hidden.

The difference between web services and data services is also obvious:

1. A data service is a particular web service which has a core XML data type that allows for easy manipulation of the return data.
2. A data service has the data retrieving, aggregating and transforming capabilities either by invoking the query engine or being the query engine.

4.6 Summary

In this chapter, we introduced two ways to realize updates in the XML-based data integration system. One is to support XQuery updates, the other is to support updates by Service Data Objects. First we listed the difficulties in updating a view. Then we introduced approaches in updating relational views of relational data, updating XML views of relational data and updating XML views in XML-based data integration system. The W3C XQuery Update Facility 1.0 was introduced. The four basic concepts of Service Data Objects were proposed including data objects, data graph, and change summary. How to realize updates by Service Data Objects in an XML-based data integration system was explained by examples. In the next chapter, we will introduce our trigger model. The trigger model should be compatible to both alternatives.

Chapter 5

Enforcing Global Constraints and Triggers

In Chapter 4 we discussed the updates in the XML-based data integration systems. When the XML-based data integration systems support updates, how to ensure the data consistency in the whole data integration system must be settled. This chapter will go to this problem. We consider to enforce the data consistency among the data sources in the data integration systems by triggers. So we will begin with the trigger model in the data integration system. Then we will introduce how to use the triggers to enforce data consistency in the data integration systems. The architecture of the XQuery trigger service will be given. Then we will go to details including how to detect event, how to schedule triggers in the conflict set, how to evaluate conditions, how to generate actions, and so on. Trigger termination and failure handling problems will follow. We will present the trigger analysis problem at the end.

5.1 Trigger Model in Data Integration System

Trigger model includes trigger meta model and trigger execution model. The trigger meta model defines the Event, Condition and Action parts of the trigger [VSCR00]. The data model of the data integration system defines the data structure and the operations that can be executed over data. Conditions and actions are specified considering the data model of the data integration system. Hence, they may implicitly concern several data sources in the data integration system.

Normally a trigger consists of three components: the triggering operation, the trigger condition and the trigger action. A trigger is *triggered* when one of its triggering operations occurs; it is *being considered* when its condition is under evaluation; it is *executed* when its action is performed. The definition of an event specifies the set of event types representing the significant situations that have to be observed across and within the data integration system and the way such events have to be detected, composed and notified.

Triggers are executed according to an execution model that characterizes the coupling of event consumption, condition evaluation and action execution within and across data integration system. Implementing the above models does not demand to re-define the data integration systems and their applications completely. We isolate the active mechanism from the data integration system control.

5.2 XQuery Trigger Model for XML-DIS

5.2.1 Design of Trigger for XML-DIS

We assume that active rules in the XML-based data integration systems follow the ECA paradigm, that is, they are triggered by specific events, include a declarative condition and a sequence of procedural actions. The event part contains a primitive event, which is restricted to instant or periodic temporal events, insertions, deletions, and qualified updates on specific attributes or properties. The condition part contains a boolean expression of predicates. Predicates can be either simple comparisons between terms (variables or constants), or the special predicates which compare newly inserted and deleted tuples (old values and new values). The trigger model should permit the users to write arbitrarily complex predicates. The action part contains a sequence of commands.

5.2.2 XQuery Trigger

As mentioned in Chapter 4, XQuery updates are not supported by XML-based data integration systems until now. We try to make our model extendable. Our design goal is that once the XML-based data integration system supports ad hoc XQuery Updates proposed by W3C in the future, our XQuery trigger model can still be used.

We decided to adopt a slightly modified trigger model based on Active XQuery, due to its simplicity and good compatibility with XQuery. Our XQuery trigger uses the update syntax in conformance to the W3C standard update facility [W3C08b] and adheres to the spirit of SQL99, which has gained tremendous popularity for developing data-intensive applications. The meta model of the XQuery trigger is defined in Figure 5.1. We extend the uniform constraint model in Chapter 3. The extensions mainly include: first, we define the trigger name in the second line. Second, in the action part, we permit INSERT, DELETE, REPLACE and even external operations. In the uniform constraint model, action part is mainly "Pop Up Error Message".

1:	DECLARE NAMESPACE ns1 (,ns)*,.	Declare which data objects are related in the trigger
2:	CREATE TRIGGER Trigger-Name	Trigger name
3:	ON (INSERT|DELETE|REPLACE)+	Trigger-associated operations
4:	OF XPathExpression (,XPathExpression)*	Trigger-relative elements
5:	[FOR EACH (NODE|STATEMENT]	Trigger granularity
6:	[XQuery-Let-Clause]	Defines the XQuery variables
7:	[WHEN XQuery-Where-Clause]	Trigger Condition
8:	DO	Trigger Action
	INSERT VALUES INTO XPathExpression|	
	DELETE XPathExpression|	
	REPLACE XPathExpression with VALUES |	
	ExternalOP|	
	ERROR Message	

Fig. 5.1 XQuery Trigger Model

We list the most distinctive capabilities of the trigger model in the following:

1. Namespace. In order to fit for the XML-DIS, the data objects which are related in the trigger are declared in the namespace definition using their global names.

2. Granularity: FOR EACH NODE/STATEMENT. A node-level XQuery trigger executes once for each node of the affected nodes. A statement-level trigger executes once for each set of nodes of the affected node. Note that the elements of a node at the global level might come from multiple data sources. The execution will take place in multiple data sources in this case. Default is node-level.
3. Transition values. If the trigger is node-level, variable OLD_NODE and NEW_NODE denote the affected element in the XML view in its before and after state. If the trigger is statement-level, variable OLD_NODES and NEW_NODES denote the sequence of affected elements in the XML view of the underlying data sources in its before and after state. The execution model must provide access to the old and new values.
4. Complexity of predicates. Arbitrarily complex XPath and XQuery predicates should be permitted. An XQuery engine is thus needed to parse them.
5. Action part. It can invoke procedures that are executed remotely and accept XML-formatted parameters (e.g., according to the SOAP protocol). Compared to the uniform constraint model in Chapter 3, more actions are permitted in the action part.

The advantages by enforcing active rules in XML-based data integration systems through XQuery triggers include: firstly, active rules defined by XQuery triggers are automatically executed after global updates on the underlying data sources are detected as events and the evaluation of the condition is true. There is no need to introduce an explicit coordination mechanism or a workflow description responsible for invocation. Secondly, the action part of XQuery triggers may invoke procedures that are executed remotely. This permits the rapid integration of an e-service based on XQuery triggers with other e-services implemented with traditional technologies. Thirdly, XQuery trigger, due to its simplicity, permits the active rules to be rapidly prototyped and tested.

5.2.3 Examples

The major usage of triggers in running applications is integrity constraint checking, referential integrity constraint checking, logging and maintenance of derived data. Our examples try to cover these areas. We use relational database and XML files as assumed data sources since they represent most of the data sources in an XML-based data integration system. The name with postfix “DB” means the underlying data source is a relational database while the name with postfix “XML” means the underlying data source is an XML file. Other data sources, such as files, web applications, can be defined similarly. Our examples are tested in the BEA AquaLogic 3.0 for WebLogic 8.1. Therefore, “ld:DataServices” appears in the namespace declaration. [1]

5.2.3.1 Integrity Constraints.

Example 5.2.1 is defined above two data sources: CustomerOrderLineItemDB and ProductXML.xml. It checks that the selling price in CustomerOrder-LineItemDB must be greater than the price set by the manufacturer in ProductXML.xml. Since this trigger is defined above two data sources, we declare two namespaces at the beginning (Line 1,2). When there is an “INSERT” in CustomerOrderLineItemDB (Line 4), that means, the seller wants to sell a product and decide the price of it, the trigger will consult the ProductXML.xml to get the price decided by the manufacturer (Line 5,6,7). The trigger condition part (Line 8) compares selling price with manufacturer price. If the selling price is less than the manufacturer price, the trigger will pop up an error message (Line 9). The “INSERT” will not be executed and the user will know the reason of the rejection.

By the UPDATE operation we can just replace the “INSERT” in line 6 with “UPDATE”.

Example 5.2.1 is quite similar with Example 3.4.2 in Chapter 3. The difference is that here we use “CREATE TRIGGERNAME”.

[1] “ld” is the abbrievation of LiquidData, which is an predecessor product of AquaLogic from BEA.

```
1: DECLARE NAMESPACE
2:   ns1 = ld:DataServices
              /CustomerOrderLineItemDB/CustomerOrderLineItem
     ns2 = ld:DataServices
              /ProductXML/PRODUCTRECORD
3:  CREATE TRIGGER SellPriceGreaterThanListPriceConstraint
4:  ON INSERT OF ns1:CustomerOrderLineItem()
                          /CUSTOMER_ORDER_LINE_ITEM
5:  LET $newLine = NEW_NODE/CUSTOMER_ORDER_LINE_ITEM
6:  LET $id := $newLine/PROD_ID
7:  LET $price := ns2:PRODUCTRECORD()/PRODUCTRECORD
              /PRODUCT[PRODUCT_ID=$id]/LIST_PRICE
8:  WHEN $newLine/PRICE < $price
9:  DO popuperrormessage('Sell price must be
                              higher than manufacturer price')
```

Example 5.2.1: Selling Price Greater than Manufacturer Price

5.2.3.2 Referential Constraint: Insert Checking.

Suppose the customer information is in the file CustomerXML.xml and the customer order is in CustomerOrderDB. Suppose the Customer_ID in CustomerOrderDB refers to the Customer_ID in CustomerXML.xml. When a new customer order is inserted, the data integration system will check whether the customer exists. This is a referential constraint at the global level and can be defined by an XQuery trigger as shown in Example 5.2.2. Line 1 defines two namespaces representing two data sources: one is the CustomerOrder in CustomerOrderDB, the other is the CUSTOMERRECORD in CustomerXML. Line 2 defines the trigger name. Line 3 defines that the trigger should be checked when there is an INSERT operation on table CUSTOMER_ORDER of the first data source. Line 4 defines three variables which will be used in the condition part and the action part. The first variable is the new inserted order. The second variable is the customerID($cid) in the new inserted order. The third variable should consult the second data source to get the customer information according to the customerID ($cid). Line 5 judges whether such customer information in the second data source exists. If there does not exist such a customer with this Customer_ID, the trigger will popup an error message. (Action part, Line 6).

By the UPDATE operation we can just replace the “INSERT” in line 6 with “UPDATE”.

```
1: DECLARE NAMESPACE
      ns1 = ld:DataServices/CustomerOrderDB/CustomerOrder
      ns2 = ld:DataServices/CustomerXML/CUSTOMERRECORD
2: CREATE TRIGGER CustomerMustExistConstraint
3: ON INSERT OF ns1:CustomerOrder/CUSTOMER_ORDER
4: LET $newOrder = NEW_NODE/CUSTOMER_ORDER
   LET $cid := $newOrder/C_ID
   LET $customer :=ns2:CUSTOMERRECORD()
                /CUSTOMERRECORD/CUSTOMER[CUSTOMER_ID=$cid]
5: WHEN fn:not(not($customer))
6: DO popuperrormessage('Customer must exist
                               when an order is inserted!')
```

Example 5.2.2: : Referential Constraint: Insert Checking

5.2.3.3 Referential Constraint: Cascade Delete.

Suppose there are two databases: CustomerOrderDB and CustomerOrderItemDB. In CustomerOrderDB, the customer order information is stored, e.g. the shipping address, the paying methods, etc. In CustomerOrderItemDB the items of the orders are stored. CustomerOrderItemDB.ORDER_ID refers to CustomerOrderDB.ORDER_ID. When an order in CustomerOrderDB is deleted, the items of the deleted order stored in the CustomerOrderLineDB must be deleted, too. The trigger is shown in Example 5.2.3. The explanation is similar to the explanation in Example 5.2.2.

```
1: DECLARE NAMESPACE
     ns1 = ld:DataServices/CustomerOrderDB/CustomerOrder
     ns2 = ld:DataServices/CustomerOrderItemDB/CustomerOrderLineItem
2: CREATE TRIGGER ReferentialConstraintForCustomerOrder
3: ON DELETE OF ns1:CustomerOrder()/CUSTOMER_ORDER
4: LET $id := OLD_NODE/ORDER_ID
5: FOR $orderLine in ns2:CustomerOrderLineItem()
                        /CUSTOMER_ORDER_LINE_ITEM
6: WHEN $id=$orderLine/ORDER_ID
7: DO delete node $orderLine
```

Example 5.2.3: : Referential Constraint: Cascade Delete

5.2.3.4 Logging.

Suppose the customer information is in the XML file CustomerXML.xml. Whenever a customer information is deleted, the deleted information will be logged into the relational database LoggingDB. The trigger is shown in

Example 5.2.4. The first three lines (Line 1, 2, 3) define the two namespaces. The first one is the XML file and the second one is the relational database where the deleted information will be stored. Line 4 is the trigger name. Line 5 defines the trigger operation “DELETE”. The trigger associated element is “ns1:CUSTOMERRECORD()/CUSTOMERRECORD/CUSTOMER”. Line 6 and Line 7 define the two variables which record the current data and time. Line 8 is the action part. The deleted information together with the deleting date and the time will be recorded in the relational database LoggingDB into the table LOGGING.

```
1: DECLARE NAMESPACE
2:  ns1 = ld:DataServices/CustomerXML/CUSTOMERRECORD
3:  ns2 = ld:DataServices/LoggingDB/LOGGING
4: CREATE TRIGGER LoggingTriggerForCustomer
5: ON DELETE OF ns1:CUSTOMERRECORD()/CUSTOMERRECORD/CUSTOMER
6: LET $date := fn:current-date()
7: LET $time := fn:current-time()
8: DO  insert node
   <LOGGING>
      <OBJECTID>{OLD_NODE/CUSTOMER_ID}</OBJECTID>
      <EVENT>Customer is deleted</EVENT>
      <DATE>{$date}</DATE>   <TIME>{$time}</TIME>
     </LOGGING>
     into ns2:LOGGING()
```

Example 5.2.4: : Logging Trigger

5.2.3.5 Accumulating Trigger.

Example 5.2.5 is a trigger among three data sources. Whenever a product is sold (reflected, an INSERT in the relational database CustomerOrderItemDB), the trigger will consult the ProductXML.xml to get the name of the manufacturer. Then the total sale amount of the manufacturer, which is in the relational database ManufactureSalesDB, will be increased. The first four lines (Line 1-4) define the namespaces. The first namespace is the CutomerOrderLineItem table in the CustomerOrderItemDB. The second one is the PRODUCTRECORD file in the ProductXML.xml file. The third one is the ManufactureSales table in the database ManufactureSalesDB. Line 5 defines the trigger name. Line 6 defines the trigger operation “INSERT”. Line 7 defines the trigger-associated element "ns1:CustomerOrderLineItem()/CUSTOMER_ORDER_LINE_ITEM". This means the trigger must detect the insert operation on this element. Line 8-13 define the variables which will be used in the condition part and the action part. During the variable definition, it is necessary to consult the data integration system to get the manufacture name in the ProductXML data source (Line 11) and get the current total sales in the ManufactureSales table (Line 12). Line 14 is the action part which replace the current total sales with the new value.

```
1: DECLARE NAMESPACE
2:    ns1 = ld:DataServices/CustomerOrderItemDB/CustomerOrderLineItem
3:    ns2 = ld:DataServices/ProductXML/PRODUCTRECORD
4:    ns3 = ld:DataServices/ManufactureSalesDB/ManufactureSales
5: CREATE TRIGGER CountingTrigger
6: ON INSERT OF
7: ns1:CustomerOrderLineItem()/CUSTOMER_ORDER_LINE_ITEM
8: LET $prodid := NEW_NODE/PROD_ID
9: LET $newprice := NEW_NODE/PRICE
10: LET $mount := NEW_NODE/QUANTITY
11: LET $man := ns2:PRODUCTRECORD()/PRODUCTRECORD
             /PRODUCT[PRODUCT_ID=$prodid]/MANUFACTURER
12: LET $curtotal := ns3:ManufactureSales()
                     /MANSALES[MANUFACTURER=$man]/GROSSSALES
13: LET $newtotal := $curtotal + $newprice * $mount
14: DO replace value of node
     ns3:ManufactureSales()/MANSALES[MANUFACTURER=$man]/GROSSSALES
   with $newtotal
```

Example 5.2.5: : Accumulating Trigger

By the UPDATE operation we can just replace the "INSERT" in line 6 with "UPDATE".

5.2.4 Assumptions for XQuery Triggers in XML-DIS

A rule is triggered when one of its associated events occurs. Rule processing is started whenever a rule is triggered, and consists in the iteration of rule selection and rule execution [CCF01].

We assume that rule processing initiates immediately after events are detected, and terminates when all triggered rules have been executed. There are no deferred triggers.

5.3 The Architecture of Trigger Service

5.3.1 ECA rules in Active Databases

Triggers, or ECA rules, appeared first in the active database systems. Figure 5.2 gives an abstract active rule system architecture as part of an active database system [PD99].

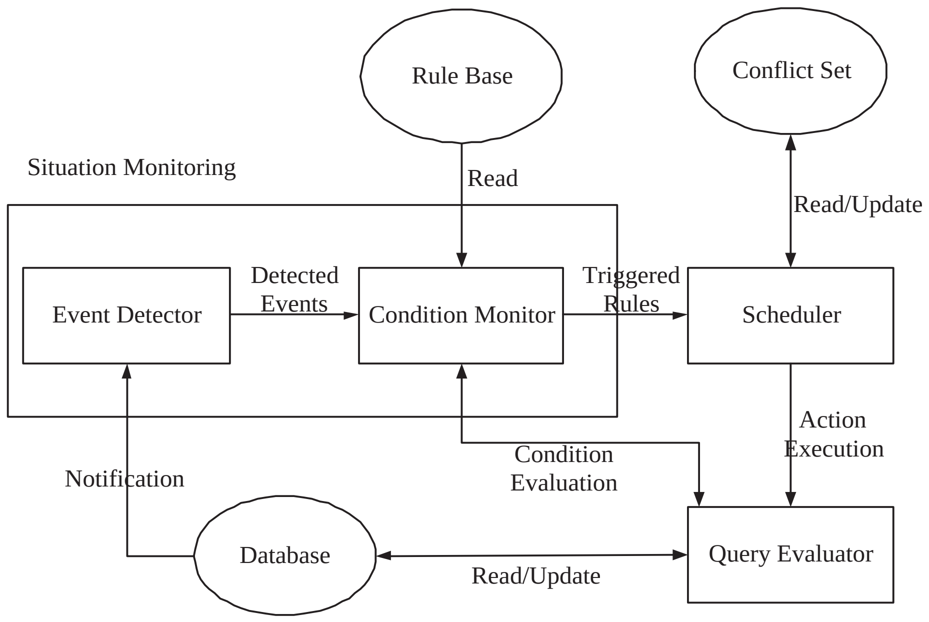


Fig. 5.2 Abstract Architecture for Active Database Systems

The principle processes are depicted in rectangles and the data sources in ellipses. Both are used to implement the functionality of an active rule system for an active database system.

The principle processes are as follows:

1. The **Event Detector** ascertains what events, which the rule system is interested in, have taken place. Primitive events are notified from the database. Composite events are constructed from incoming primitive events plus information about past events that can be obtained from the history. There exist systems that support condition-action rules. In this kind of system, there is no explicit statement of the events to be moni-

tored. But normally actual implementations do have to monitor primitive events.

2. The **Condition Monitor** evaluates the conditions of rules associated with events that have been detected by the **Event Detector**.
3. The **Scheduler** compares recently triggered rules with those that have previously been triggered, updates the conflict set, and fires any rules that are scheduled for immediate processing.
4. The **Query Evaluator** executes database queries or actions. Access may be required both to the current state of the database and to past states in order to support monitoring of how the database is evolving.

This architecture is designed for using database systems. But for a trigger service system, this reference architecture cannot be totally reused. We will introduce the trigger service system for an XML-based data integration system in the next section.

5.3.2 The Architecture

An active XML-based data integration system requires an integration-wide mechanism for event handling and action execution. In such a context it must be possible to detect events to make them visible to the trigger service of the integration system. It must be also possible to couple the execution of actions with the execution of data integration applications. Figure 5.3 shows the architecture of the trigger service in the XML-based data integration system [LM08b].

When the client application submits an update to the DIS (Step 1), the DIS will call the trigger service (Step 2). The trigger service will consult the trigger repository and fetch the related triggers (Step 3). The fetched triggers are collected in the conflict set (Step 4). Then the trigger scheduling component will fetch the triggers in the conflict set according to the scheduling rules (Step 5) and hand over to the condition evaluation component (Step 6). Then the trigger service will evaluate the condition. During condition evaluation, it is possible to query other data sources through the DIS (Step 7). An XQuery engine is used to execute the complete XQuery expression in the condition

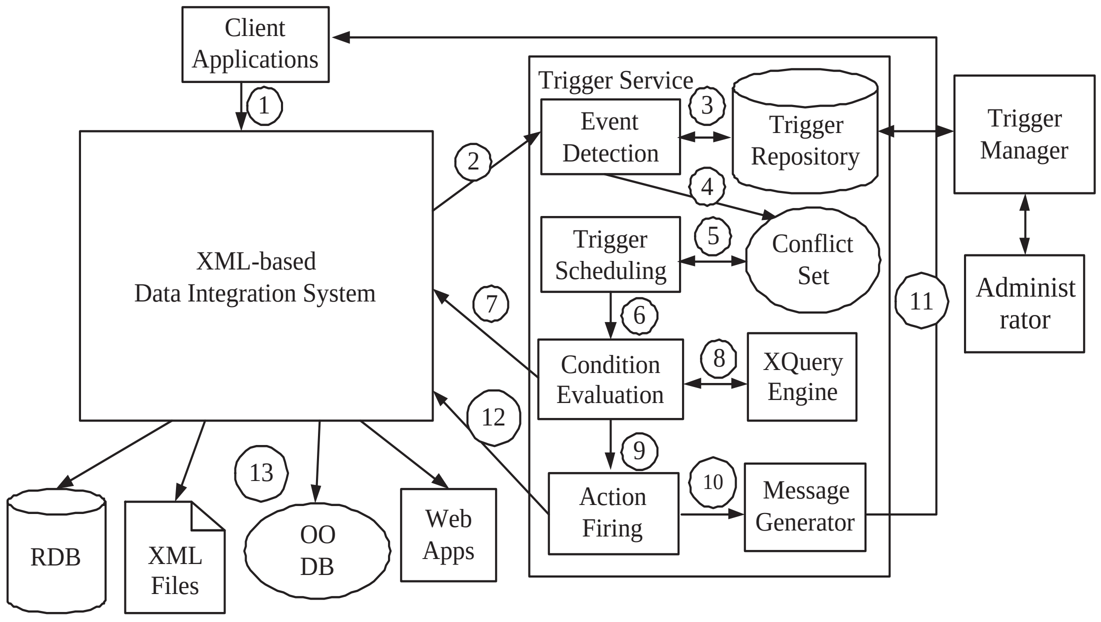


Fig. 5.3 Architecture of the XQuery Trigger Service

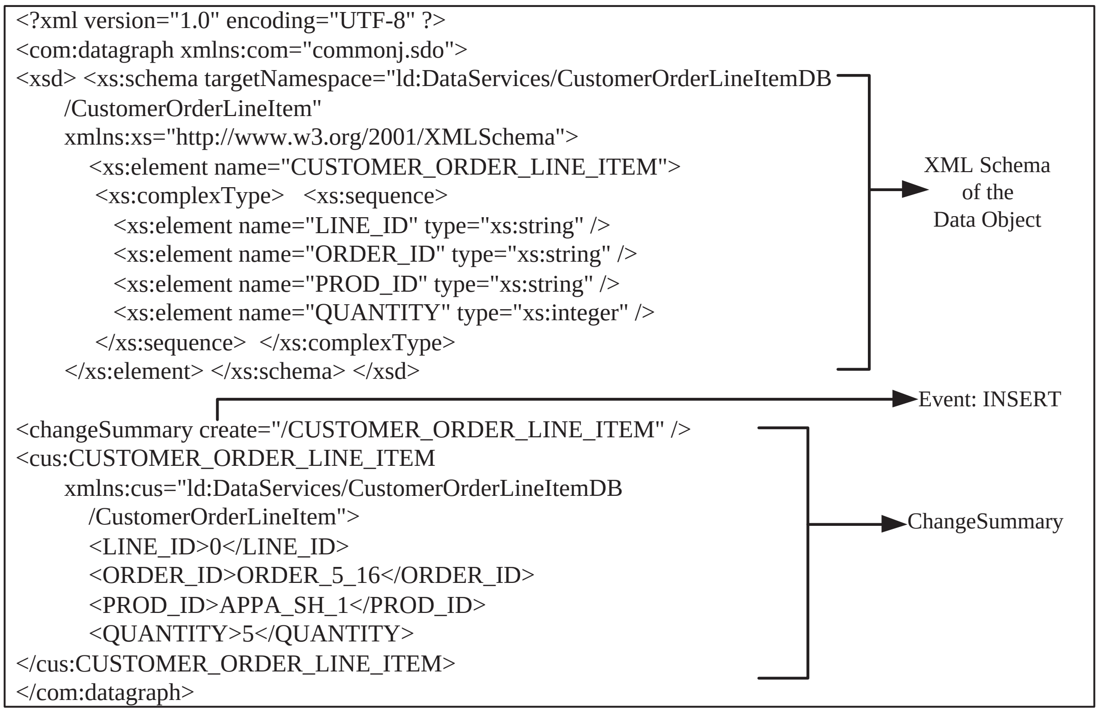
(Step 8). The evaluation result will be sent to the action firing component (Step 9). If the condition is evaluated to true, there are two possible kinds of actions: one is an error message, the other is a set of updates. If the action is an error message, it means that the triggered active rules are “CHECK” constraints and the constraints are violated. The action firing component will call the message generator (Step 10) and an error message will be sent back to the users or the client applications (Step 11). The operation will be aborted so the data integrity is guaranteed. If the actions are a set of updates, they will be sent to the DIS (Step 12) and the update will be executed in the underlying data sources (Step 13).

We will use the XQuery trigger in Example 5.2.5 as an example to explain the details of the trigger service.

5.3.3 Main Components

5.3.3.1 Event Detection

Suppose the client application submits the data graph (Step 1 in Figure 5.3) serialized in XML to the data integration system shown in Figure 5.4. Events can be detected by analyzing the ChangeSummary. The first line of ChangeSummary denotes the operation type (INSERT, DELETE, or REPLACE). The old data objects and the new data objects are listed in ChangeSummary which can be used to compare with the trigger target XPath expression and which provide the transition values.

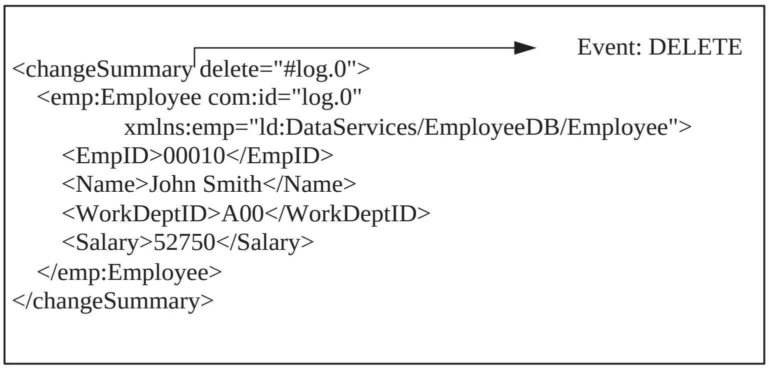


```
<?xml version="1.0" encoding="UTF-8" ?>
<com:datagraph xmlns:com="commonj.sdo">
<xsd> <xs:schema targetNamespace="ld:DataServices/CustomerOrderLineItemDB
      /CustomerOrderLineItem"
      xmlns:xs="http://www.w3.org/2001/XMLSchema">
        <xs:element name="CUSTOMER_ORDER_LINE_ITEM">
         <xs:complexType>   <xs:sequence>
           <xs:element name="LINE_ID" type="xs:string" />
           <xs:element name="ORDER_ID" type="xs:string" />
           <xs:element name="PROD_ID" type="xs:string" />
           <xs:element name="QUANTITY" type="xs:integer" />
         </xs:sequence>  </xs:complexType>
      </xs:element> </xs:schema> </xsd>

<changeSummary create="/CUSTOMER_ORDER_LINE_ITEM" />
<cus:CUSTOMER_ORDER_LINE_ITEM
      xmlns:cus="ld:DataServices/CustomerOrderLineItemDB
        /CustomerOrderLineItem">
        <LINE_ID>0</LINE_ID>
        <ORDER_ID>ORDER_5_16</ORDER_ID>
        <PROD_ID>APPA_SH_1</PROD_ID>
        <QUANTITY>5</QUANTITY>
</cus:CUSTOMER_ORDER_LINE_ITEM>
</com:datagraph>
```

Fig. 5.4 Analysis of the Submitted DataGraph with ChangeSummary

Figure 5.5 shows an example which will execute a delete. The old values are included in it.



```
<changeSummary delete="#log.0">
   <emp:Employee com:id="log.0"
              xmlns:emp="ld:DataServices/EmployeeDB/Employee">
      <EmpID>00010</EmpID>
      <Name>John Smith</Name>
      <WorkDeptID>A00</WorkDeptID>
      <Salary>52750</Salary>
   </emp:Employee>
</changeSummary>
```

Fig. 5.5 An Example of ChangeSummary which Executes DELETE

Figure 5.6 shows an example which will execute an update. The old values and the new values are included in it.

```
<!-- ChangeSummary -->
<changeSummary>                                   Event: Update
  <Customer com:ref="/Profile/Customer[1]">
    <Name>Jing Chu</Name>
  </Customer>
  <Address com:ref="/Profile/Customer[1]/Address[1]">
    <Zip>224233</Zip>
  </Address>
  <Address com:ref="/Profile/Customer[1]/Address[2]">
    <City>Tangyang</City>
  </Address>
  <Street com:ref="/Profile/Customer[1]/Address[2]/Street">
    <Name>TangxinStreet 38</Name>
  </Street>
</changeSummary>
<!-- Root Data Object -->
<ns0:Profile xmlns:ns0="ld:DataServices/CustomerProfileXML/CustomerProfile">
    <Customer>
      <CustomerID>C3457</CustomerID>
      <Name>Jing Lu</Name>
      <Address>
        <AddID>A2200</AddID>
        <Zip>70569</zip>
        ...
      </Address>
      <Address>
        <AddID>A2201</AddID>
        <Street>
          <StrID>38</StrID>
          <Name>Universitaetsstrasse</Name>
          ...
        </Street>
        ...
        <City>Stuttgart</City>
      </Address>
      ...
    </Customer>
</ns0:Profile>
```

Fig. 5.6 An Example of ChangeSummary which Executes UPDATE

By analyzing the ChangeSummary of the data graph, we can judge the event type, the change target XPath expression and get the new values and old values.

5.3.3.1.1 Primitive vs. Composite Events

There are two kinds of events: primitive and composite events. Primitive events are raised by a single low-level occurrence. Composite events are raised by some combination of primitive or composite events. Composite event handling presents challenges in terms of semantics and efficiency that have not been yet fully addressed [PD99]. In data integration systems, because the data sources are loosely-coupled, it is not necessary to combine the events in different data sources. Therefore, we consider here only the primitive events.

5.3.3.1.2 Order-Dependent vs. Order-Independent Events

The concept of event in the XML world is different from that defined for tuples in the relational model or for objects in the object-oriented models. Order is critical for XML documents. In the triggers for XML repositories, there are normally two kinds of event semantics: order-dependent and order-independent, along with the distinction between positional and non-positional operations. However, if XML is used as an interchange format (e.g., in XML-DIS), order is less relevant. Only if the trigger's data sources are all XML files, we use order-dependent semantics and permit positional operations. If the trigger's data sources are mixed (including RDB, XML files, txt files, etc) or include none XML data sources, we use order-independent event semantics and permit no positional operations.

5.3.3.1.3 Explicit vs. Derived Event Detection Model

There are normally two kinds of event detection methods: explicit and derived. Explicit events are those updates coming from XQuery Update language or updates from application code. Explicit events will be detected immediately. Derived events are normally detected by applying a delta algorithm ("XML-Diff" algorithm [BBCC02]) on the data sources whereby the history versions must be kept. We consider explicit events only since it is impossible to store the history versions of each data source in the integration level.

5.3.3.1.4 Immediate vs. Deferred Event Consumption Model

Since after the updates are submitted to the underlying data sources, control over them is lost, a deferred event consumption model needs coordination mechanism, which adds the complexity of the trigger execution model (e.g., timestamp and the global clock are then needed). We assume the immediate event consumption model.

5.3.3.2 Triggering Phase

After the events are detected (Step 2 in Figure 5.3), the trigger service will look up the trigger repository and extract the triggers that are related to the events (Step 3). These triggers are collected in the conflict set (Step 4).

5.3.3.3 Trigger Scheduling

Trigger scheduling determines what happens when multiple rules are triggered at the same time. This phase indicates how the triggers are extracted to the conflict set, how they are selected for evaluation and in which order conditions are processed. There are two principle issues which must be considered: one is the selection of the next rule to be fired, the other is the number of rules to be fired.

5.3.3.3.1 The Selection of the Next Trigger to be Fired

Trigger order can strongly influence the result. Conflict resolution policies are normally based on (1) the time of update (i.e., the time of event occurrence); or (2) the complexity of the condition [PD99].

Nowadays the active database systems tend to support priority schemes, in which triggers are associated with a static priority. Static priorities are often determined either by the system (e.g., based on trigger creation time) or by the user as an additional attribute of the trigger. In the latter case, a trigger is selected from the conflict set for execution using a priority mechanism. Triggers can be placed in order using a numerical scheme, in which each trigger is given an absolute value that is its priority [SJGP94], or by indicating the relative priorities of triggers by stating explicitly that a given trigger must be fired before another when both are triggered at the same time [ACL91].

In our work, if there are triggers whose action part is the "PopupErrorMessage" (see example in Example 5.2.1), we extract and evaluate them first because they represent integrity constraints and they should be enforced first since integrity constraint checking must be executed before any other operations. If the action part of the fired trigger is not "PopupErrorMessage", we mainly use the first policy. Triggers are selected according to the event detection time.

5.3.3.3.2 The Number of Triggers to be Fired

There are mainly two options to decide on the number of triggers to be fired, as explained in the following:

1. Fire all triggers in the conflict set sequentially;
2. Fire all triggers in the conflict set in parallel;

Which approach is most appropriate depends upon the task that is being supported by the trigger. The first option is suitable for triggers supporting integrity maintenance. It means an update is successful once all constraints have been validated. The second option can encourage more efficient trigger processing.We use the first option in our work (first-in-first-out) to decide the order of trigger execution (Step 5 in Figure 5.3). The selected triggers are sent to the condition evaluation component (Step 6).

5.3.3.4 Condition Evaluation

Most of the conditions in a trigger are only related to the OLD_NODE and NEW_NODE (normally only the changed target elements). However, it is also possible that the conditions are related to other data sources. For example, in Example 5.2.5, the trigger must query the DIS to get the name of the manufacturer stored in ProductXML.xml. In our trigger service, we permit the user to write arbitrary conditions. An XQuery Engine is needed to evaluate the condition. Before starting the XQuery engine, notations such as NEW_NODE, OLD_NODE, and queries to the DIS (Step 7 in Figure 5.3) will be replaced with actual values because the XQuery engine cannot parse these foreign notations. A complete XQuery FLWOR expression is built and sent to the Engine (Step 8). The evaluation result is sent to the action firing component (Step 9).

We will interpret each step of the working procedure of the XQuery Engine in the following paragraphs.

5.3.3.4.1 Interpreting **LET** Clause

The value of a variable defined in this **LET** clause can be an atomic value (such as a String or an Integer value) or an XML node (an XML fragment). The value can come from a constant, OLD_NODE or NEW_NODE, a build-in function, another variable, or a query to the data integration system [LM09].

The keyword OLD_NODE and NEW_NODE in the **LET** clause can not be understood by the XQuery engine. They must be replaced with actual values before being sent to the XQuery engine. Old value and new value of a modified target are stored as a string with atomic value or an XML fragment form. Hence, they can be directly used to replace the OLD_NODE and NEW_NODE. For the variable whose value comes from evaluation of other data sources, queries to the data integration system must be called first. An XML fragment, or an array of XML fragments will be returned. When all of the foreign notations are replaced with actual values, the condition part can ultimately be accepted by the XQuery engine.

5.3.3.4.2 Interpreting **FOR** Clause

The XQuery **FOR** clause in the trigger iterates over an input sequence. We restrict the input sequence to the values obtained by calling a data service's read function. The **FOR** clause has three forms as shown in Example 5.3.1.

The variable in a **FOR** clause can be considered as iterating over an array returned by evaluating the XPath expression that contains the data service's read function. For instance, the XPath expression in **Line 1** returns all "Cus-

```
1:  FOR $cus in ns0:CustomerProfile()/Profile/Customer
2:  FOR $add in ns0:getCustomerByID(C3456)/Profile
        /Customer/Address
3:  FOR $name in ns0:CustomerProfile()/Profile
        /Customer[CustomerID=C3456]/Address/Street/Name
```

Example 5.3.1: Some examples of **FOR** clause in constraint or trigger

tomer" from the "CustomerProfile" query to the data integration system. The returned data objects form an XML fragment array. The variable "cus" iterates over this array. **Line 2** returns "Address" from the "Customer" whose id is "C3456". **Line 3** returns all street name(s) under the address(es) of Customer "C3456".

5.3.3.4.3 Processing *CHECK* Constraints

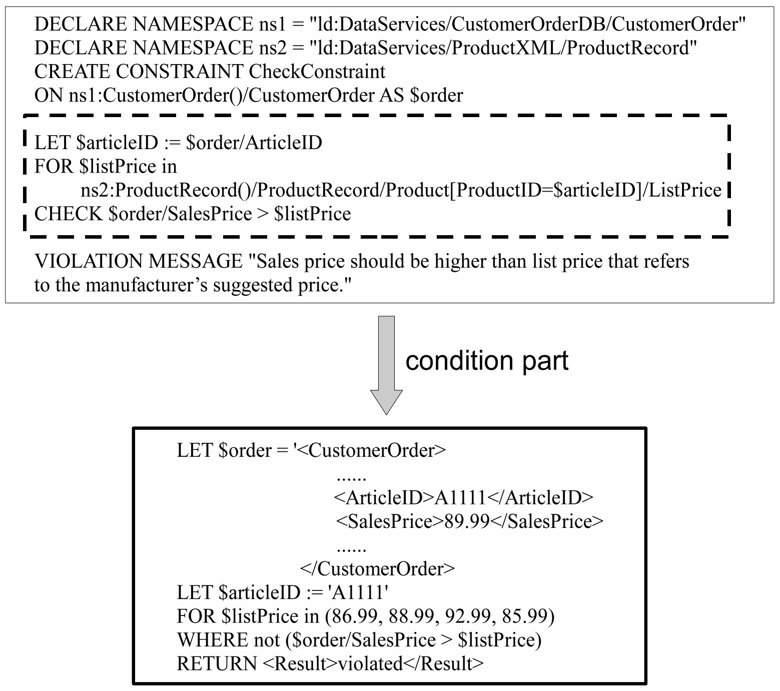
If the action part of a trigger is PopupErrorMessage, we call this trigger a **CHECK** constraint. In order to build a complete XQuery expression, a self-defined return clause is appended at the end of the condition part. Meanwhile, a logical negation is performed on the logical expression in the *CHECK* constraint. The complete XQuery expression for the condition part of check constraint is shown in Example 5.3.2.

```
XQuery-Let-For-Clauses
WHERE not (XQuery-Logical-Expression)
RETURN <Result>violated</Result>
```

Example 5.3.2: : Building a Complete XQuery FLWR-Clause for CHECK Constraint

We use Figure 5.7 to illustrate the process of building the complete XQuery expression from the condition part of a constraint.

In this example, the "Product" element in data service "ProductRecord" might have multiple "ListPrice" elements. Hence, we can use a *FOR* clause to iterate all list prices of a product. It can be seen from Figure 5.7 that after the processing by using algorithms described above, every variable is assigned an actual value. The variable "order" is assigned the new value of the monitored target after modification, namely an XML fragment of "CustomerOrder". "articleID" is assigned the article id of the new value. The



```
DECLARE NAMESPACE ns1 = "ld:DataServices/CustomerOrderDB/CustomerOrder"
DECLARE NAMESPACE ns2 = "ld:DataServices/ProductXML/ProductRecord"
CREATE CONSTRAINT CheckConstraint
ON ns1:CustomerOrder()/CustomerOrder AS $order
LET $articleID := $order/ArticleID
FOR $listPrice in
      ns2:ProductRecord()/ProductRecord/Product[ProductID=$articleID]/ListPrice
CHECK $order/SalesPrice > $listPrice
VIOLATION MESSAGE "Sales price should be higher than list price that refers
to the manufacturer's suggested price."
```

```
LET $order = '<CustomerOrder>
                     ......
                     <ArticleID>A1111</ArticleID>
                     <SalesPrice>89.99</SalesPrice>
                     ......
                 </CustomerOrder>
LET $articleID := 'A1111'
FOR $listPrice in (86.99, 88.99, 92.99, 85.99)
WHERE not ($order/SalesPrice > $listPrice)
RETURN <Result>violated</Result>
```

Fig. 5.7 Building a Complete XQuery Expression

XPath expression in the *FOR* clause is replaced with a sequence of atomic values, which represent four list price suggested by four different manufacturers. Then, the logical negation operation is performed on the logical expression in the *CHECK* clause and the *CHECK* clause is converted to an XQuery-Where-Clause. Finally, a self-defined return clause is appended at the end of the condition part to construct a complete XQuery expression. The complete XQuery expression will be sent to the XQuery engine to be evaluated. If any item in the *FOR* sequence does not satisfy the condition in the *WHERE* clause, the XML element <Result>violated</Result> will be returned. A single return would mean that the whole constraint is violated. In this example, the constraint is violated by the performed modification, because there is a list price "92.99" which is greater than the sales price "89.99".

5.3.3.4.4 Processing Trigger

A complete XQuery expression must also be built to represent the condition part of trigger. The form of the complete XQuery expression for the condition part of a trigger is shown in Example 5.3.3.

```
    XQuery-Let-For-Clauses
    WHERE XQuery-Logical-Expression
    RETURN <Result>fired</Result>
```

Example 5.3.3: : Building a Complete XQuery FLWR-Clause for Trigger

Once an XML element <Result>fired</Result> is returned, that means the trigger is fired and the defined actions should be performed. The generation of the whole XQuery expression is the same as processing the CHECK constraints.

5.3.3.5 Action Firing

If the action part is an error message, the trigger service will call the message generator (Step 10 in Figure 5.3) and send error message back to the client application (Step 11). The operation will be aborted, so the data integrity is guaranteed. If the action part is a set of updates, the trigger service will map the updates into corresponding data graphs serialized in XML. These data graphs will be sent to the DIS (Step 12) and the update will be executed in the data sources (Step 13). Figure 5.8 shows this procedure.

In our example, when a pair of shoes is sold, the trigger in Example 5.2.5 will be fired. The trigger service will consult the data integration system to get the name of the manufacturer of the shoes, which is stored in ProductXML.xml. Then the XQuery trigger service will generate a data graph with ChangeSummary serialized in XML, as shown in Figure 5.9. This action will update the "GROSSSALES" in ManufactureSalesDB from 999.95 to 1999.99.

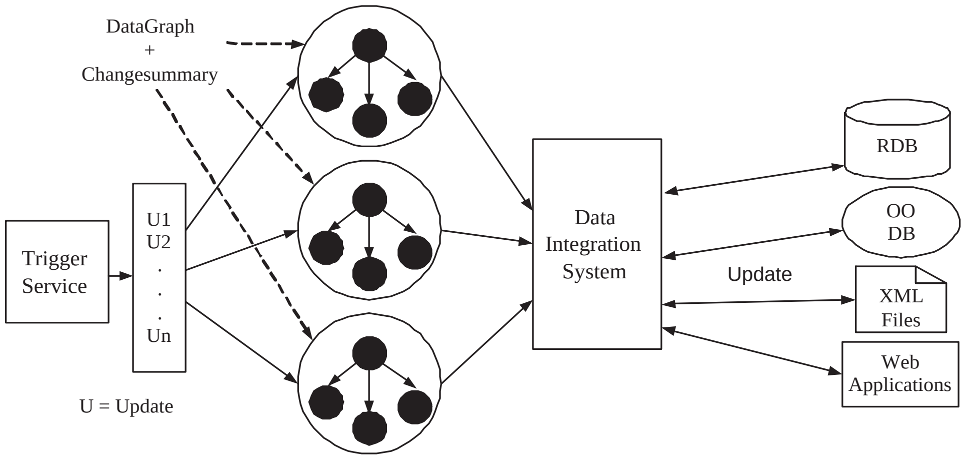
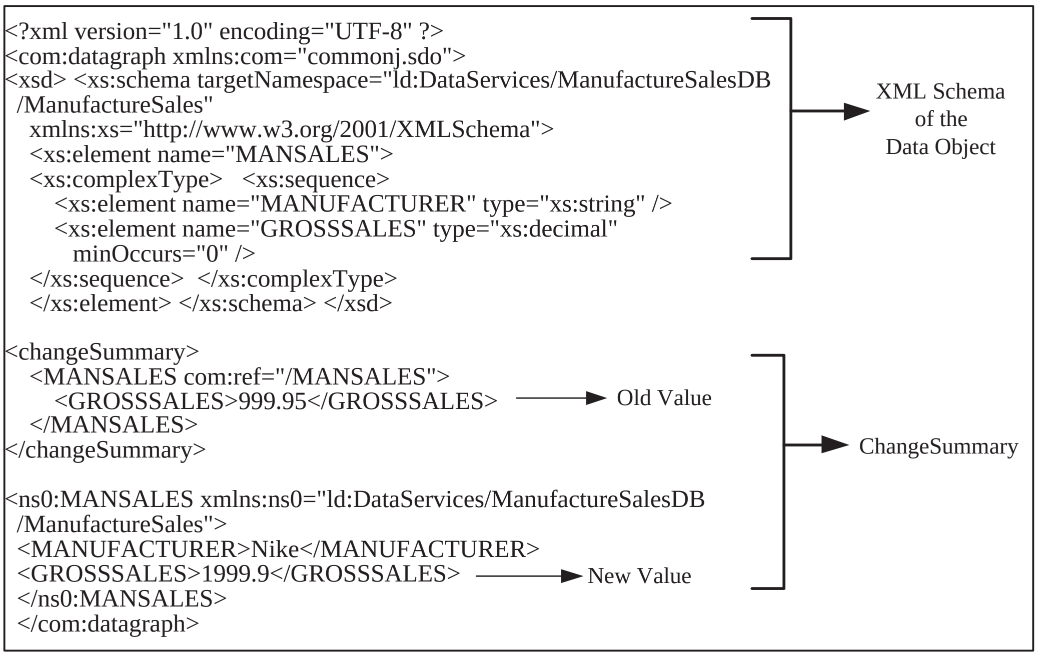


Fig. 5.8 Action Firing, DataGraph Generating and Update Execution for Step 8 in Figure 5.3



```
<?xml version="1.0" encoding="UTF-8" ?>
<com:datagraph xmlns:com="commonj.sdo">
<xsd> <xs:schema targetNamespace="ld:DataServices/ManufactureSalesDB
 /ManufactureSales"
  xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="MANSALES">
  <xs:complexType>   <xs:sequence>
    <xs:element name="MANUFACTURER" type="xs:string" />
    <xs:element name="GROSSSALES" type="xs:decimal"
      minOccurs="0" />
  </xs:sequence>  </xs:complexType>
  </xs:element> </xs:schema> </xsd>

<changeSummary>
  <MANSALES com:ref="/MANSALES">
    <GROSSSALES>999.95</GROSSSALES>
  </MANSALES>
</changeSummary>

<ns0:MANSALES xmlns:ns0="ld:DataServices/ManufactureSalesDB
 /ManufactureSales">
 <MANUFACTURER>Nike</MANUFACTURER>
 <GROSSSALES>1999.9</GROSSSALES>
 </ns0:MANSALES>
 </com:datagraph>
```

XML Schema of the Data Object
Old Value
ChangeSummary
New Value

Fig. 5.9 Generated DataGraph with ChangeSummary

5.3.3.6 Trigger Termination

It is possible that during action execution some actions will trigger new actions. Thus a cycle gets formed and the trigger execution cannot be terminated. If there are no condition parts in the triggers, this can be avoided by applying the DAG (Directed Acyclic Graph) algorithm when the trigger is added to the repository. But if there are condition parts, it becomes impossible to decide whether there is a cycle in trigger execution or not, because the condition evaluation result depends on the content of the changed target or the data sources which might change unpredictably. Some work goes a further step to analyze the trigger behaviour in the XML repository [BPW01].

In our work if a given trigger executes an operation and this in turn causes additional triggering, the trigger execution context is suspended, and a new procedure is recursively invoked. The depth of the recursion is limited by a system-specific threshold. If the depth is over that threshold, an exception will be thrown. For a simple and direct execution of the trigger service in our work, we set the threshold as two. This means that we only permit t_i fires t_j and t_j is not permitted to fire any trigger. Thus it is impossible to form a circle and the trigger termination is guaranteed. Trigger termination problem is difficult and some issues on how to analyze triggers will be discussed in Section 5.5.

5.3.3.7 Failure Handling

It is also possible that the execution of actions will come across failures. There are two kinds of failures. One is simply called a metric failure, which means that an action cannot be finished in a limited timebound. This can be caused when the data sources are overloaded. Another failure is called a logic failure, which is caused by catastrophic failure of the underlying data sources [CGMW96]. In our system, failures are detected and flagged. The client application can thus know the failure causes.

5.4 Implementation and Evaluation

We use BEA AquaLogic Dataservice Platform 3.0 for WebLogic 8.1 as the XML-based data integration system [BEA08a] which supports SDO programming framework. The application server is the BEA WebLogic Server 8.1 [BEA08b]. We use Software AG's Tamino XML Server [AG06] as the repository where XQuery triggers are stored. We also use the Tamino's XQuery engine to parse XQuery expressions. Refer to Figure 5.3 again, we also implement a trigger manager, which is a graphical interface for the administrators to add, delete, change and query the triggers. Experiments show that the whole system runs very well [Wan08] and can fulfill the requirements of enhancing the data consistency in the data integration systems. Compared with the traditional method of implementing active rules by programming code, our method has the following advantages [GSS04]:

1. Heterogeneous data sources can interact easily through XQuery triggers in the data integration systems. There is no need to know the location, the characteristics, the interfaces of the data sources participating in the integration.
2. XQuery triggers enable a uniform description of the active rules and integrity constraints relevant to the data integration systems. In contrast, when active rules and integrity constraints are embedded into application programs, they can be specified and implemented in different ways and in several applications. The correct enforcement of business rules depends on the reliability of programmers, which might cause severe inconsistencies.
3. A stand-alone XQuery trigger service provides a way to ensure that every application obeys the same rules and avoids redundant integrity constraint checking.
4. Using XQuery triggers facilitates readability and maintenance of business rules. They can be inserted, maintained and managed easily in the repository. There is no need to go through programming code, which saves a great deal of effort in developing active rules.

5.5 Trigger Analysis

Trigger analysis deals with predicting how a set of triggers behaves at runtime. The following are the three properties of trigger behaviour in active database systems (ADBs) [PD99]:

- **Termination.** ADB triggers are terminating only if there is no recursive firing of triggers.

- **Confluence.** Confluence property of triggers decides whether the execution order of non-prioritized triggers makes any difference in the final database state.
- **Observable Determinism.** A trigger set is observably deterministic, if the effect of trigger processing as observed by the user of the system is independent of the order in which the fired triggers are selected for processing.

The following two triggers implement a response, which is confluent but observably nondeterministic.

```
On <event E1>
if <condition C1>
do <send message to user>

On <event E1>
if <condition C1>
do <abort>
```

In this example, if the first trigger is scheduled for firing before the second, then the user receives a message and the transaction is aborted. By contrast, if the second trigger is scheduled before the first, then the transaction is aborted but no message is sent to the user. But the state of the database does not change if we change the execution order of the two triggers.

The key analysis question is the *termination* of the trigger execution. A set of triggers is said to be *terminating* if for any initial event and any initial database state, the trigger execution terminates. Triggering and activation relations between triggers have been used to determine whether a set of triggers is terminating.

A trigger r_i may trigger a trigger r_j if the action of r_i may generate an event which triggers r_j. [AWH92] [AHW95] proposed the triggering graph which represents each trigger as a vertex, and there is a directed arc from a vertex r_i to a vertex r_j if r_i may trigger r_j. Acyclicity of the triggering graph implies definite termination of trigger execution.

[BW94] used an activation graph which also represents triggers as vertices. In this case an arc between two distinct vertices r_i and r_j indicates that r_j's condition may be changed from false to true after the execution of r_i's action, while an arc from a vertex r_i to itself indicates that r_i's condition may be true after the execution of r_i's action. Acyclicity of this graph also implies definite termination of trigger execution.

Triggering and activation graphs were combined in [BCP95] in a method called rule reduction which gives more precise results than either of the previous methods. With the method, any vertex which does not have both an incoming triggering and activation arc can be removed from the graph, along with its outgoing arcs. This removal of vertices is repeated until there are no such vertices. If the procedure results in all the vertices being removed, then the trigger set is definitely terminating.

Unfortunately, until now there is no universal rule to ensure the termination of triggers. But it is now widely recognized that user interactions are necessary and sometimes sufficient during the analysis of the triggers.

5.6 Summary

This chapter presented an approach to realize an XQuery trigger service in an XML-based data integration system. We specified a trigger model based on an XQuery trigger that is in conformance to XQuery Update model of W3C and we defined the semantics of the model. This model was also similar to the uniform constraint model proposed in Chapter 3. The difference was that the trigger model in this chapter focused on the action part which was much complicated than the constraint model and where the triggers could execute various data operations. We explained also how to write constraints and triggers using this XQuery trigger model by examples. Then the architecture of the whole XQuery trigger service was proposed. Main components including event detection, trigger scheduling, condition evaluation, action firing were discussed. How to ensure the termination of the trigger execution and how to handle failures were also discussed. XQuery trigger provided a direct and universal tool for assigning data consistency enforcement rules. We built up the whole prototype under an SDO programming framework and presented the detailed execution model. Experiments were referenced and showed that the XQuery trigger service could fulfill the requirements of enhancing data consistency in XML-based data integration systems.

Chapter 6

Related Work

Using constraints and triggers to enhance XML-based data integration systems is related to three aspects of related work, the first one is semantic query optimization, the second one is data integrity, and the third one is active systems. This chapter compares our work with related work in these three aspects.

6.1 Semantic Query Optimization

In [SO89], a user specified query is optimized by describing a scheme to utilize semantic knowledge. The semantic is represented as function-free clauses in predicate logic. The scheme uses a graph theoretic approach to identify redundant joins and restrictions in a given query. An optimization algorithm is presented which eliminates redundant nonprofitable specifications. Relationships among entity schema, semantics and query form the basis of the algorithm. The similarity between this work and ours is that both algorithms are based on schema, semantics and query. The difference is that this work uses subset constraints and implication constraints, while we use integrity constraints. Another difference is that this work must use the index information of a database, while we do not rely on indexes, because indexes are not always available in data sources.

An implementation of two semantic query optimization techniques, predicate introduction and join elimination, in DB2 universal database is described in [CGK+99]. The experiments show that SQO can lead to dramatic performance improvements. Only referential constraints and check constraints are used. This work is targeted to DB2 and ours is to XML-based DIS. In addition, it depends on the indexing mechanism of DB2, while again in a data integration system indexes are not always available.

An efficient semantic query optimization algorithm, which runs best when the database is large or when the query execution cost is expected to be high, is presented in [PLO91]. The transformation process classifies the predicates into *imperative, optional, or redundant.* At the end of the transformation process, all the *imperative* predicates are retained, while the *redundant* predicates are eliminated. *Optional* predicates are retained or discarded based on the estimated cost and benefit of retaining them. Again this work is for object-oriented DBMS and ours is for XML-based data integration system. This optimizer is embedded in the DBMS and ours is used by the data integration system at the service level, for example, on the web through a browser.

A system focusing on semantic query optimization for query plans of heterogeneous multi-database systems is presented in [HK00]. This approach reduces the cost of query plans generated by an information mediator by providing necessary and sufficient conditions to eliminate unnecessary joins in a conjunctive query of arbitrary join topology. The difference between this work and ours is that the optimization is targeted to a query plan generated by the mediator of the data integration system, while ours is targeted to the query generated by the user of the data integration system. This optimizer must be embedded in the query mediator while ours is built on top of the query mediator and thus provides better extendibility and flexibility. However, the elimination techniques are also applicable in our approach as well.

An intelligent system using tool-supported techniques to optimize mediated queries are proposed by [BBM01]. The techniques rely on the availability of integration knowledge including: local source schemata, a virtual mediated schema and its mapping descriptions. These three kinds of integration knowledge form the base of the query reformulation rules, while in our approach we rely on the integrity constraints.

6.2 Data Integrity

[DD95] [VA96] [CGMW96] [TC97] [GW93] address the problem of integrity constraint checking in multi-database environments. However, what they consider are tightly-coupled distributed databases in which global transactions, global queries, and global concurrency control are present.

In [CGMW96] the authors have presented a framework and a toolkit for constraint management in loosely-coupled heterogeneous environments. The framework allows the formal specification of the interface each database offers, and of the constraint management strategies. Then the formal guarantees regarding the consistency of constraints can be proved. The framework can express guarantees that are more "relaxed" than in conventional database systems, in the sense that constraints may hold only at given times or under certain conditions. This added flexibility is essential in real-world distributed scenarios where it is not possible to guarantee that integrity constraints are always satisfied. The toolkit provides configurable constraint management and translation processes, and a library of proven strategies, making it relatively easy to enforce relaxed constraints. We do not permit the relaxed constraints in our work because relaxed constraints normally need a global clock, so that the constraints can be checked at the given time referring to the global clock.

In the framework each information source chooses an interface which can offer to the local constraint manager (LCM) for each of its data items involved in a multi-source constraint. The interface specifies how the data item may be read, written, and/or monitored by the LCM. Applications inform the constraint manager (CM) of constraints that need to be monitored or enforced. Based on the constraint and the interfaces available for the items involved in the constraint, the CM decides on the constraint management strategy it executes. This strategy monitors or enforces the constraint. The degree to which each constraint is monitored or enforced is formally specified by the guarantee. In our work, we do not necessarily have a local constraint manager, because we want to ensure the local autonomy of the data sources in the data integration system.

Interfaces are specified using a notation based on events and rules. It is assumed that the interfaces for the data items involved in constraints are specified by a “constraint administrator” at each site, based on the level of access and performance that can be provided to the CM for the data item. The strategy for a constraint describes the algorithm used by the CM to monitor or enforce the constraint. Like interfaces, strategies are specified using a notation based on events and rules. In addition to performing operations on the data items involved in a constraint, strategies may evaluate predicates over the values of data items (obtained through read operations) and over private data maintained by the CM. Constraints in our work are defined at the global level by the data integration system administrators. If the administrators of the local data sources want the constraints from the local data sources also be checked at the global level, they can also submit the local constraints to the constraint service. The constraint wrapper will translate the constraints into the uniform constraint model and the translated constraints will be stored in the constraint repository. Remember that before being stored into the constraint repository, the administrator of the data integration system needs to give an authentication in order to decide whether the constraints can be checked at the global level.

[VA96] presents an approach to database interoperation that exploits the semantic information provided by integrity constraints defined on the component databases. The authors identify two roles of integrity constraints in database interoperation: (1) a set of integrity constraints describing valid states of the integrated view can be derived from the constraints defined on the underlying databases. (2) local integrity constraints can be used as a semantic check on the validity of the specification of the integrated view. The ideas are illustrated in the context of an instance-based database interoperation paradigm, where objects rather than classes are the unit of integration. Two notions of *objectivity* and *subjectivity* are introduced as an indication of whether a constraint is valid beyond the context of a specific database, and the impact of these notions are demonstrated.

The object comparison rules are introduced in the database interoperation. The condition part of object comparison rules, like object constraints, imposes conditions which should be satisfied by object instances. Two types of object comparison rule conditions are distinguished by the authors: (1) Inter-object conditions. These are conditions involving both objects to be compared. (2) Intra-object conditions. There are conditions on a simple object. Compared with our work, inter-object conditions are similar to the global constraints defined above two or more data sources. Intra-object conditions are similar with global constraints, which are defined on only one data source.

In [TC97], the authors propose a flexible way to realize global integrity maintenance in federated database systems. It aims at integrating the active rule paradigm into a federated database framework. Thereby, integrity constraints involving multiple component database systems can easily be supported. Active rules provide a powerful mechanism for communication and cooperation between heterogeneous database systems. In addition, they are used for specifying enforcement of constraints in case of violation.

At the federated level, global active mechanisms are offered in form of global ECA rules. These ECA rules are stored in the global database which also contains all data of the federation layer. According to the event part, every global ECA rules is attached to a global rule manager which is responsible for the management and processing of this ECA rule. For global integrity enforcement global ECA-rules are not sufficient, because global ECA-rules alone cannot prevent local applications from performing operations which may have an effect on the global integrity. Therefore, local ECA-rules and local rule managers are needed. Local ECA-rules and local rule managers reside in a mediator layer on top of the local component databases. Local ECA rules are used to specify actions which are performed upon the occurrence of a local event. Like global ECA rules, local ECA-rules are managed and processed by a local rule manager. A local rule manager detects a local event in the component database and informs the appropriate global rule manager about the event occurrence. In contrast to global ECA-rules, local ECA-rules are stored locally at the component system level. The only purpose of these local ECA-rules is to signal the occurrence of a specific local event to the federation layer. In a data integration system, the data sources are more heterogeneous, while in the federated database system the components are all databases. Data sources such as files, XML files, web applications cannot signal the occurrence of an event to the data integration system. Therefore, in our work, we consider only the events from the global updates. Local updates to the local data sources are not considered.

[GW93] presents an optimization for integrity constraint verification in distributed databases. The optimization allows a global constraint, i.e. a constraint spanning multiple databases, to be verified by accessing data at a single database, eliminating the cost of accessing remote data. The optimization is based on an algorithm that takes as input a global constraint and data to be inserted into a local database. The algorithm produces a local condition such that, if the local data satisfies this condition, then based on the previous satisfaction of the global constraint, the global constraint is still satisfied. If the local data does not satisfy the condition, then a conventional global verification procedure is required. The optimization in this work is executed in the database systems. Both global constraints and local constraints are in the same form. But in a data integration system, the local constraints are different from one data source to another and are also different from the global constraints. It is hard to reuse this optimization, not to say that there are data sources in the data integration system which do not even have their own constraint mechanisms.

[GW97] describes a family of cooperative constraint checking protocols for federated database systems. Many of the constraint checking protocols here are also applicable to local systems that do not support transactions. However, this work depends on the message mechanism, which can be supported by the underlying database systems, but cannot be ensured by the data sources in a data integration system, which are more heterogeneous.

6.3 Active Systems, ECA Rules, and Triggers

6.3.1 ECA Rules for XML

[BPW02] [BPW01] [BCP01] [BBCC02] [PPW03] propose new languages for defining ECA rules on XML, providing reactive functionality on XML repositories. The language is based on reasonably expressive fragments of the

XPath [CSD99] and XQuery standards. The difference is that the ECA rules in these proposals are applied to XML repositories, while our ECA rules are applied to the XML views of the underlying data sources.

[BPW02] [BPW01] propose a new language for defining ECA rules on XML, providing reactive functionality on XML repositories, and develop new techniques for analysing the triggering and activation dependencies between rules defined in this language. The language is based on reasonably expressive fragments of the XPath and XQuery standards.

[BBCC02] proposes Active XQuery, an active language for XML repositories that is based on a previously defined XQuery Update Model in [TIHW01]. In particular, it presents the syntax and semantics of the Active XQuery language, aiming at emulating the trigger definition and execution model of SQL3.

[PPW03] describes a language for ECA rules on XML and a prototype implementation of this language. They also discuss some preliminary ideas regarding a language for ECA rules operating on a graph/triple representation of RDF, and describe the architecture of a distributed deployment of such RDF ECA rules.

[IO01] proposes and validates XBML (Xml-based Business Modeling Language) as an XML active query language approach to specify electronic commerce business models. This work is for distributed XML databases while ours is for XML data integration system.

6.3.2 Distributed Active Information Systems

[ABM04] proposes Active XML (AXML) which is a declarative framework that harnesses web services for data integration and is put to work in a peer-to-peer architecture. In each peer, a web service engine, an AXML service call execution engine and an XML database as peer repository must be installed. AXML can also be used to define interaction rules among peers. But it does not fit data integration systems, because it is impossible to install these engines and the XML database in the legacy data sources.

FRAMBOISE [FGD97] proposes a construction system for ECA-services decoupled from a particular DBMS. Event detection is performed by event detectors which have to be specialized for the respective DBMS. A rule service is responsible for the maintenance of the rule base and implements rule execution. This work is designed for only one database while ours is for a data integration system.

[SNS05] [SNS06] propose an active system whereby users can place triggers on immaterialized nested XML views of relational data. In this architecture, the authors present scalable and efficient techniques for processing triggers over nested views by leveraging existing support for SQL triggers over flat relations in commercial relation databases. The performance results indicate that the proposed techniques are a feasible approach to support triggers over nested views of relational data. In this approach triggers are based on Active XQuery. This work is for relational data while ours is for a wide range of underlying data sources.

[VSCR00] presents a service-based architecture that provides flexible and independent active capabilities suitable for federated database system applications. Rule and event services are proposed to cooperate to specify and to execute ECA rules. They respectively offer flexible rule execution and event management adapted to different participating DBMS characteristics. CoopWARE [MGKS96] and TriggerMan [HK97] also propose execution mechanisms for distributed rules. The former has been developed for a workflow environment, and the latter proposes an asynchronous trigger processor as an extension module for an object-relational DBMS. All the data sources here are database systems while data sources in our work are in a wider range.

[CL98] proposes an architecture of a framework to support ECA rules suitable for distributed and heterogeneous systems. It uses an expressive composite event specification language: Snoop. The action part of the ECA rules is written in a combination of OQL and C++. Our model adheres to the SQL99 [Mel03] trigger spirit and uses the W3C Update standard model which we believe has won more popularity and is more uniform.

[GETE88] presents an approach to integrate relational databases with support for global updates. The proposed approach is suitable for situations where users need to be explicitly aware of storage locations for data items, as well as for situations where storage locations are determined by attribute values. In our work the users of the active rules do not have to explicitly know the storage locations. They only have to know the global names of the data objects.

[AAC$^+$99] proposes a specification language for active view definition above an XML repository while our work is above an XML data integration system. The active views are mapped into local functions, while the action parts of triggers in our work are mapped into data graphs.

Chapter 7

Conclusion

Data integration has long been recognized as a very important problem in the database and information system research field. The appearance of the Web has motivated the research on XML-based data integration systems. In XML-based data integration systems, the user submits an XQuery and gets back the result in XML. After the XML-based data integration system accepts the query from the user, it will parse it, divide it, push down the query plan to the wrappers of the data sources. The wrappers of the data sources will translate the sub-queries into the local query language. The sub-queries will be executed in the data sources and the wrappers will translate the result back to XML. The XML-based data integration system will combine the results from each data source and return the final result in XML to the user.

Users are always expecting high query performance and data consistency from the data integration system. This work goes a step further in satisfying these requirements from the users by supporting constraints and triggers in the XML-based data integration system. Constraints and triggers have played an important role in database systems both in improving the query capability and enforcing data consistency. Constraints from the databases can be used to optimize the queries semantically. Triggers can be used in the traditional database systems to ensure data consistency among tables. Based on these ideas, this work discussed two main usage scenarios of constraints and triggers in an XML-based data integration system. The first usage is to use the constraints from the data sources and global referential constraints to semantically optimize the queries submitted to the data integration system. The second is to ensure the global data consistency in the data integration system by enforcing global constraints and triggers.

In order to use the constraints from the local data sources to semantically optimize the queries, we first define a uniform constraint model to express the inherently heterogeneous constraints from the different data sources. The uniform constraint model is based on an XQuery trigger model. We develop a constraint wrapper, which borrows the concept of the integration wrappers in the data integration system. A constraint wrapper can translate the heterogeneous constraints from the data sources into the uniform constraint model. After the translation, the constraints coming from the data sources are expressed in the uniform form. Then these constraints are stored in the constraint repository. We developed a semantic query optimizer for the data integration system. When the user submits a query to the data integration system, the query will be transformed to our semantic query optimizer. We implemented three semantic query optimization techniques in the optimizer including detection of empty result, join elimination and predicate elimination using the constraints in the constraint repository. The main idea of the "detection of empty result" is that if the query condition is in conflict with the constraints, the query will return an empty result. Therefore, the empty result will be detected before the query is further processed by the data integration system. Thus a great deal of processing resource is saved. The main idea of "join elimination" is that if there is a join in the query condition which is supported by a referential constraint in the constraint repository then this join will always hold true. Under some circumstances, the join will be eliminated by our optimizer and thus there is no need to scan two data sources. Therefore, a great deal of communication costs and processing resources are saved. The main idea of "predicate elimination" is that if a predicate in the query condition is subsumed by the constraints in the constraint repository, this predicate will always hold true and thus can be eliminated. Again this saves a great deal of communication and processing cost. We analyzed the effects of the optimizations and concluded that our optimizer works best when the data volume is huge and the query cost is supposed to be high. Especially when the data sources are non-relational databases, which are quite common in an XML-based data integration system, our optimizer works best.

On the other hand, more and more XML-based data integration systems announced to support update at the global level either by built-in functions or by programming frameworks. Among these methods Service Data Objects programming is the most uniform and is proposed by most of the industry-leading vendors and open-source communities. We realized the global update under the Service Data Objects programming framework. When the data integration system supports updates at the global level, data consistency among the data sources could be enhanced. In order to enforce the data consistency at the global level in the XML-based data integration system, a global trigger model is defined. We used again an XQuery trigger model whose action part can include all kinds of data manipulation operations and even external operations. Compared to the uniform constraint model, the XQuery trigger model focuses on what actions should be taken when the triggers are fired. The namespace declaration is used to define which data objects are participating in the global trigger and constraint. We gave examples to show how to define the global constraints and triggers using the XQuery trigger model. Then we proposed the architecture to support an XQuery trigger service in the data integration system in order to enhance data consistency for the whole data integration system. Important components such as event detection, trigger scheduling, condition evaluation, action firing and trigger termination are discussed. By enforcing the data consistency rules with triggers, the heterogeneous data sources can interact with each other at the global level. The XQuery trigger service enables a uniform definition of data consistency enforcement rules and guarantees a central management of the data consistency enforcement rules and thus ensures that all applications in the data integration systems conform to the same constraint and trigger enforcement rules. By using the XQuery trigger service, the XML-based data integration system becomes active rather than passive in a sense that it can take some actions when the specified events are detected and the corresponding conditions are evaluated to be true.

Appendix A
XML Schema of Data Graph Serialization

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:sdo="commonj.sdo"
   targetNamespace="commonj.sdo">
    <xsd:element name="datagraph" type="sdo:DataGraphType"/>
    <xsd:complexType name="DataGraphType">
    <xsd:complexContent>
       <xsd:extension base="sdo:BaseDataGraphType">
          <xsd:sequence>
            <xsd:any minOccurs="0" maxOccurs="1"
             namespace="##other" processContents="lax"/>
            </xsd:sequence>
          </xsd:extension>
    </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType name="BaseDataGraphType" abstract="true">
        <xsd:sequence>
          <xsd:element name="models" type="sdo:ModelsType" minOccurs="0"/>
          <xsd:element name="xsd" type="sdo:XSDType" minOccurs="0"/>
          <xsd:element name="changeSummary" type="sdo:ChangeSummaryType"
            minOccurs="0"/>
        </xsd:sequence>
     <xsd:anyAttribute namespace="##other" processContents="lax"/>
      </xsd:complexType>
       <xsd:complexType name="ModelsType">
        <xsd:annotation>
          <xsd:documentation>
            Expected type is emof:Package.
          </xsd:documentation>
        </xsd:annotation>
        <xsd:sequence>
            <xsd:any minOccurs="0" maxOccurs="unbounded"
            namespace="##other" processContents="lax"/>
        </xsd:sequence>
        </xsd:complexType>

        <xsd:complexType name="XSDType">
           <xsd:annotation>
           <xsd:documentation>
                 Expected type is xsd:schema.
           </xsd:documentation>
           </xsd:annotation>
           <xsd:sequence>
             <xsd:any minOccurs="0" maxOccurs="unbounded"
             namespace="http://www.w3.org/2001/XMLSchema"
             processContents="lax"/>
           </xsd:sequence>
          </xsd:complexType>

         <xsd:complexType name="ChangeSummaryType">
         <xsd:sequence>
         <xsd:any minOccurs="0" maxOccurs="unbounded"
          namespace="##any"
           processContents="lax"/>
         </xsd:sequence>
         <xsd:attribute name="create" type="xsd:string"/>
         <xsd:attribute name="delete" type="xsd:string"/>
         <xsd:attribute name="logging" type="xsd:boolean"/>
         </xsd:complexType>
         <xsd:attribute name="ref" type="xsd:string"/>
</xsd:schema>
```

References

[AAC$^+$99] Serge Abiteboul, Bernd Amann, Sophie Cluet, Adi Eyal, Laurent Mignet, and Tova Milo. Active views for electronic commerce. In *VLDB '99: Proceedings of the 25th International Conference on Very Large Data Bases*, pages 138–149, San Francisco, CA, USA, 1999. Morgan Kaufmann Publishers Inc.

[ABM04] Serge Abiteboul, Omar Benjelloun, and Tova Milo. Positive active xml. In *PODS '04: Proceedings of the twenty-third ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 35–45, New York, NY, USA, 2004. ACM.

[ACL91] Rakesh Agrawal, Roberta Cochrane, and Bruce G. Lindsay. On maintaining priorities in a production rule system. In *VLDB '91: Proceedings of the 17th International Conference on Very Large Data Bases*, pages 479–487, San Francisco, CA, USA, 1991. Morgan Kaufmann Publishers Inc.

[ACPS96] S. Adali, K. S. Candan, Y. Papakonstantinou, and V. S. Subrahmanian. Query caching and optimization in distributed mediator systems. *SIGMOD Rec.*, 25(2):137–146, 1996.

[AG06] Software AG. Number one in xml management: Tamino xml server, technical factsheet, 2006.

[AG09] Software AG. Xml data management, 2009. http://www.softwareag.com/corporate/products/wm/tamino/default.asp.

[AHW95] Alexander Aiken, Joseph M. Hellerstein, and Jennifer Widom. Static analysis techniques for predicting the behavior of active database rules. *ACM Trans. Database Syst.*, 20(1):3–41, 1995.

[AQM$^+$97] Serge Abiteboul, Dallan Quass, Jason McHugh, Jennifer Widom, and Janet L. Wiener. The Lorel query language for semistructured data. *International Journal on Digital Libraries*, 1(1):68–88, 1997.

[ASV06] Serge Abiteboul, Luc Segoufin, and Victor Vianu. Representing and querying xml with incomplete information. *ACM Trans. Database Syst.*, 31(1):208–254, 2006.

[AWH92] Alexander Aiken, Jennifer Widom, and Joseph M. Hellerstein. Behavior of database production rules: termination, confluence, and observable determinism. In *SIGMOD '92: Proceedings of the 1992 ACM SIGMOD international conference on Management of data*, pages 59–68, New York, NY, USA, 1992. ACM.

[BBCC02] A. Bonifati, D. Braga, A. Campi, and S. Ceri. Active xquery. In *ICDE '02: Proceedings of the Eighteenth International Conference on Data Engineering*, pages 403–412, San Jose, USA, 2002. IEEE Computer Society.

[BBM01] D. Beneventano, S. Bergamaschi, and F. Mandreoli. Extensional knowledge for semantic query optimization in a mediator based system. In *International Workshop on Foundations of Models for Information Integration*, 2001.

[BBNP03] John Beatty, Stephen Brodsky, Martin Nally, and Rahul Paul. Next-generation data programming: Service data objects. *A Joint Whitepaper with IBM and BEA.*, 2003.

[BCF$^+$07] S. Boag, D. Chamberlin, M. Fernandez, D. Florescu, J. Robie, and J.Simeon. Xquery 1.0: An xml query language, 2007. http://www.w3.org/TR/xquery/.

[BCP95] Elena Baralis, Stefano Ceri, and Stefano Paraboschi. Improving rule analysis by means of triggering and activation graphs. In *RIDS '95: Proceedings of the Second International Workshop on Rules in Database Systems*, pages 165–181, London, UK, 1995. Springer-Verlag.

[BCP01] Angela Bonifati, Stefano Ceri, and Stefano Paraboschi. Active rules for xml: A new paradigm for e-services. *The VLDB Journal*, 10(1):39–47, 2001.

[BDH04] Vanessa P. Braganholo, Susan B. Davidson, and Carlos A. Heuser. On the updatability of xml views over relational databases. In *International Workshop on the Web and Databases (WebDB)*, San Diego, CA, USA, 2004.

[BEA03] BEA. Bea liquiddata for weblogic, building queries and data views, 2003. http://edocs.bea.com/liquiddata/docs81/querybld/index.html.

[BEA08a] BEA. Bea aqualogic data services platform 3.0, 2008. http://edocs.bea.com/aldsp/docs30/index.html.

[BEA08b] BEA. Bea weblogic server and weblogic express 8.1 documentation, 2008. http://edocs.bea.com/wls/docs81/index.html.

[BFSW01] Peter Buneman, Wenfei Fan, Jérôme Siméon, and Scott Weinstein. Constraints for semistructured data and xml. *SIGMOD Rec.*, 30(1):47–54, 2001.

[BIO$^+$] BEA, IBM, Oracle, Primeton Technologies, Rogue Wave Software, SAP, Software AG, Sun Microsystems, Xcalia, and Zend Technologies. *SDO for Java Specification V2.1.* http://www.osoa.org/download/attachments/36/Java-SDO-Spec-v2.1.0-FINAL.pdf?version=1.

[Bon02] Angela Bonifati, editor. *Reactive Services for XML Repositories.* Dissertation, 2002.

[BPW01] James Bailey, Alexandra Poulovassilis, and Peter T. Wood. Analysis and optimization for event-condition-action rules on xml. *Computer Networks*, 2001.

[BPW02] James Bailey, Alexandra Poulovassilis, and Peter T. Wood. An event-condition-action language for xml. In *WWW '02: Proceedings of the 11th international conference on World Wide Web*, pages 486–495, New York, NY, USA, 2002. ACM.

[BS81] F. Bancilhon and N. Spyratos. Update semantics of relational views. *ACM Trans. Database Syst.*, 6(4):557–575, 1981.

[BSKW91] Thierry Barsalou, Niki Siambela, Arthur M. Keller, and Gio Wiederhold. Updating relational databases through object-based views. *SIGMOD Rec.*, 20(2):248–257, 1991.

[BW94] Elena Baralis and Jennifer Widom. An algebraic approach to rule analysis in expert database systems. Technical report, Stanford, CA, USA, 1994.

[Car06] Michael Carey. Data delivery in a service-oriented world: the bea aqualogic data services platform. In *SIGMOD '06: Proceedings of the 2006 ACM SIGMOD international conference on Management of data*, pages 695–705, New York, NY, USA, 2006. ACM.

[CCF01] Fabio Casati, Silvana Castano, and Mariagrazia Fugini. Managing workflow authorization constraints through active database technology. *Information Systems Frontiers*, 3(3):319–338, 2001.

[CCW00] Stefano Ceri, Roberta Cochrane, and Jennifer Widom. Practical applications of triggers and constraints: Success and lingering issues (10-year award). In *VLDB '00: Proceedings of the 26th International Conference on Very Large Data Bases*, pages 254–262, San Francisco, CA, USA, 2000. Morgan Kaufmann Publishers Inc.

[CD83] Armin B. Cremers and G. Doman. Aim - an integrity monitor for the database system ingres. In *VLDB '83: Proceedings of the 9th International Conference on Very Large Data Bases*, pages 167–170, San Francisco, CA, USA, 1983. Morgan Kaufmann Publishers Inc.

[CDTW00] Jianjun Chen, David J. DeWitt, Feng Tian, and Yuan Wang. Niagaracq: A scalable continuous query system for internet databases. In Weidong Chen, Jeffrey F. Naughton, and Philip A. Bernstein, editors, *Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data, May 16-18, 2000, Dallas, Texas, USA*, pages 379–390. ACM, 2000.

[CFPT94] Stefano Ceri, Piero Fraternali, Stefano Paraboschi, and Letizia Tanca. Automatic generation of production rules for integrity maintenance. *ACM Trans. Database Syst.*, 19(3):367–422, 1994.

[CGK$^+$99] Qi Cheng, Jarek Gryz, Fred Koo, T. Y. Cliff Leung, Linqi Liu, Xiaoyan Qian, and K. Bernhard Schiefer. Implementation of two semantic query optimization techniques in db2 universal database. In *VLDB '99: Proceedings of the 25th International Conference on Very Large Data Bases*, pages 687–698, San Francisco, CA, USA, 1999. Morgan Kaufmann Publishers Inc.

[CGM90] Upen S. Chakravarthy, John Grant, and Jack Minker. Logic-based approach to semantic query optimization. *ACM Trans. Database Syst.*, 15(2):162–207, 1990.

[CGMH+97] S. Chawathe, H. Garcia-Molina, J. Hammer, K. Ireland, Y. Papakonstantinou, J. Ullman, and J. Widom. The tsimmis project: Integration of heterogeneous information sources. *Journal of Intelligent Information Systems*, 8(2):117–132, 1997.

[CGMW96] Sudarshan S. Chawathe, Hector Garcia-Molina, and Jennifer Widom. A toolkit for constraint management in heterogeneous information systems. In *ICDE '96: Proceedings of the Twelfth International Conference on Data Engineering*, pages 56–65, Washington, DC, USA, 1996. IEEE Computer Society.

[CL98] Sharma Chakravarthy and Roger Le. Eca rule support for distributed heterogeneous environments. In *ICDE '98: Proceedings of the Fourteenth International Conference on Data Engineering*, page 601, Washington, DC, USA, 1998. IEEE Computer Society.

[CP84] Stavros S. Cosmadakis and Christos H. Papadimitriou. Updates of relational views. *J. ACM*, 31(4):742–760, 1984.

[CSD99] J. Clark and W3C Recommendation S. DeRose, Editor. Xml path language (xpath) version 1.0, 1999. http://www.w3.org/TR/xpath.

[CW91] Stefano Ceri and Jennifer Widom. Deriving production rules for incremental view maintenance. In *VLDB '91: Proceedings of the 17th International Conference on Very Large Data Bases*, pages 577–589, San Francisco, CA, USA, 1991. Morgan Kaufmann Publishers Inc.

[CW92] Stefano Ceri and Jennifer Widom. Production rules in parallel and distributed database environments. In *VLDB '92: Proceedings of the 18th International Conference on Very Large Data Bases*, pages 339–351, San Francisco, CA, USA, 1992. Morgan Kaufmann Publishers Inc.

[CW93] Stefano Ceri and Jennifer Widom. Managing semantic heterogeneity with production rules and persistent queues. In *VLDB '93: Proceedings of the 19th International Conference on Very Large Data Bases*, pages 108–119, San Francisco, CA, USA, 1993. Morgan Kaufmann Publishers Inc.

[CX01] Josephine Cheng and Jane Xu. Xml and db2. In *ICDE '01: Proceedings of the 16th International Conference on Data Engineering*, Heidelberg, Germany, 2001. IEEE Computer Society.

[Dat08] DataDirect. Xquery for data integration, 2008. http://www.datadirect.com/products/xquery/data-integration/index.ssp.

[Dat09] DataDirect. Using xquery, 2009. http://www.xquery.com/tutorials-/xquery_tutorial/.

[DB82] Umeshwar Dayal and Philip A. Bernstein. On the correct translation of update operations on relational views. *ACM Trans. Database Syst.*, 7(3):381–416, 1982.

[DD95] Lyman Do and Pamela Drew. Active database management of global data integrity constraints in heterogeneous database environments. In *ICDE '95: Proceedings of the Eleventh International Conference on Data Engineering*, pages 99–108, Washington, DC, USA, 1995. IEEE Computer Society.

[DFF$^+$98] A. Deutsch, M. F. Fernandez, D. Florescu, A. Y. Levy, and D. Suciu. "XML-QL: A Query Language for XML". In *WWW The Query Language Workshop (QL)*, Cambridge, MA, , 1998.

[DG97] Oliver M. Duschka and Michael R. Genesereth. Answering recursive queries using views. In *PODS '97: Proceedings of the sixteenth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems*, pages 109–116, New York, NY, USA, 1997. ACM.

[DSD08] Dsd, 2008. http://www.brics.dk/DSD/.

[EC75] Kapali P. Eswaran and Donald D. Chamberlin. Functional specifications of subsystem for database integrity. In Douglas S. Kerr, editor, *Proceedings of the International Conference on Very Large Data Bases, September 22-24, 1975, Framingham, Massachusetts, USA*, pages 48–68. ACM, 1975.

[eXi08] eXist. Open source native xml database, 2008. http://exist.sourceforge.net/.

[FGD97] Hans Fritschi, Stella Gatziu, and Klaus R. Dittrich. Framboise – an approach to construct active database mechanisms. Technical report, 1997.

[Fou06] The Apache Software Foundation. Apache tomcat 6.0, 2006. http://tomcat.apache.org.

[FRV96] Daniela Florescu, Louiqa Raschid, and Patrick Valduriez. Answering queries using OQL view expressions. In *Workshop on Materialized Views, in cooperation with ACM SIGMOD*, pages 84–90, Montreal, Canada, 1996.

[FTS00] Mary Fernández, Wang-Chiew Tan, and Dan Suciu. Silkroute: trading between relations and xml. *Comput. Netw.*, 33(1-6):723–745, 2000.

[GETE88] M. Samy Gamal-Eldin, G. Thomas, and R. Elmasri. Integrating relational databases with support for updates. In *DPDS '88: Proceedings of the first international symposium on Databases in parallel and distributed systems*, pages 202–209, Los Alamitos, CA, USA, 1988. IEEE Computer Society Press.

[GETE89] M. S. Gamal-Eldin, G. Thomas, and R. Elmasri. Local and global constraints in database integration. In *Proceedings of the Twenty-Second Annual Hawaii International Conference on System Sciences, Software Track*, pages 604–611, Kailua-Kona, HI, USA, 1989. MCS Dept., Clarkson Univ., Potsdam, NY.

[Gro07] Object Management Group. Mof 2.0/xmi mapping, version 2.1.1, 2007. http://www.omg.org/docs/formal/07-12-01.pdf.

[GSS04] D. Goldin, S. Srinivasa, and V. Srikanti. Active databases as information systems. In *Proceedings of Internatioanl Database Engineering and Applications Symposium*, pages 123–130. IEEE Computer Society, 2004.

[GW93] Ashish Gupta and Jennifer Widom. Local verification of global integrity constraints in distributed databases. *SIGMOD Rec.*, 22(2):49–58, 1993.

[GW97] Paul Grefen and Jennifer Widom. Protocols for integrity constraint checking in federateddatabases. *Distrib. Parallel Databases*, 5(4):327–355, 1997.

[Hal01] Alon Y. Halevy. Answering queries using views: A survey. *The VLDB Journal*, 10(4):270–294, 2001.

[HK97] Eric N. Hanson and Samir Khosla. An introduction to the triggerman asynchronous trigger processor. In *Lecture Notes In Computer Science; Vol. 1312, Proceedings of the Third International Workshop on Rules in Database Systems table of contents*, pages 51–66. Springer-Verlag, 1997.

[HK00] Chun-Nan Hsu and Craig A. Knoblock. Semantic query optimization for query plans of heterogeneous multidatabase systems. *Knowledge and Data Engineering*, 12(6):959–978, 2000.

[IBM08] IBM. Service oriented architecture: Soa, 2008. http://www-306.ibm.com/software/solutions/soa/.

[IFF$^+$99] Zachary G. Ives, Daniela Florescu, Marc Friedman, Alon Levy, and Daniel S. Weld. An adaptive query execution system for data integration. In *SIGMOD '99: Proceedings of the 1999 ACM SIGMOD international conference on Management of data*, pages 299–310, New York, NY, USA, 1999. ACM.

[IO01] H. Ishikawa and M. Ohta. An active web-based distributed database system for e-commerce. In *In Proc. Web Dynamics Workshop*, London, 2001.

[JCV84] Matthias Jarke, Jim Clifford, and Yannis Vassiliou. An optimizing prolog front-end to a relational query system. *SIGMOD Rec.*, 14(2):296–306, 1984.

[Kel86] A. M. Keller. The role of semantics in translating view updates. *Computer*, 19(1):63–73, 1986.

[KW96] Chung T. Kwok and Daniel S. Weld. Planning to gather information. In *13th AAAI National Conf. on Artificial Intelligence*, pages 32–39, Portland, Oregon, 1996. AAAI / MIT Press.

[LC00] Dongwon Lee and Wesley W. Chu. Constraints-preserving transformation from XML document type definition to relational schema. In *International Conference on Conceptual Modeling / the Entity Relationship Approach*, pages 323–338, 2000.

[Len02] Maurizio Lenzerini. Data integration: a theoretical perspective. In *PODS '02: Proceedings of the twenty-first ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 233–246, New York, NY, USA, 2002. ACM.

[LM07] Jing Lu and Bernhard Mitschang. Dis-cs: Improving enterprise data integration by constraint service. In *ISCA 20th International Conference on Computer Applications in Industry and Engineering (CAINE 2007)*, San Francisco, California USA, 2007.

[LM08a] Jing Lu and Bernhard Mitschang. A constraint-aware query optimizer for web-based data integration. In *Proceedings of the Fourth International Conference on Web Information Systems and Technologies (WEBIST 2008)*, Funchal, Madeira, Portugal, 2008.

[LM08b] Jing Lu and Bernhard Mitschang. An xquery-based trigger service to bring consistency management to data integration systems. In *iiWAS'08: Proceedings of the 10th International Conference on Information Integration and Web-based Applications and Services*, pages 154–161, Linz, Austria, 2008. ACM Press.

[LM09] Jing Lu and Bernhard Mitschang. Enforcing data consistency in data integration systems by xquery trigger service. *International Journal of Web Information Systems*, 5(2), 2009.

[LRO96] Alon Y. Levy, Anand Rajaraman, and Joann J. Ordille. Querying heterogeneous information sources using source descriptions. In *VLDB '96: Proceedings of the 22th International Conference on Very Large Data Bases*, pages 251–262, San Francisco, CA, USA, 1996. Morgan Kaufmann Publishers Inc.

[Mar08a] MarkLogic, 2008. http://www.marklogic.com/.

[Mar08b] MarkLogic. Marklogic server technical data sheet, 2008. http://www.marklogic.com/resources/marklogic-server-technical-data-sheet.html.

[Mar08c] MarkLogic. Xml repository, 2008. http://www.marklogic.com/capabilities/xml-repository.html.

[Mel03] Jim Melton, editor. *Advanced SQL: 1999, Understanding Object-Oriented and Other Advanced Features.* Morgan Kaufmann, 2003.

[Mel05] Roman B. Melnyk. Db2 basics: An introduction to the sql/xml publishing functions, 2005. http://www.ibm.com/developerworks/db2/library/techarticle/dm-0511melnyk/.

[Men97] Elliott Mendelson, editor. *Introduction to Mathematical Logic, Fourth Edition.* Chapman & Hall/CRC, 1997.

[MGKS96] John Mylopoulos, Avigdor Gal, Kostas Kontogiannis, and Martin Stanley. A generic integration architecture for cooperative information systems. In *COOPIS '96: Proceedings of the First IFCIS International Conference on Cooperative Information Systems*, page 208, Washington, DC, USA, 1996. IEEE Computer Society.

[Mic09] Microsoft. Optimistic concurrency (ado.net data services framework), 2009. http://msdn.microsoft.com/en-us/library/cc668770.aspx.

[MUR09] Makoto MURATA. Relax ng, 2009. http://relaxng.org/.

[OB04] Mourad Ouzzani and Athman Bouguettaya. Query processing and optimization on the web. *Distrib. Parallel Databases*, 15(3):187–218, 2004.

[OCKM06] F. Özcan, D. Chamberlin, K. Kulkarni, and J.-E. Michels. Integration of sql and xquery in ibm db2. *IBM Syst. J.*, 45(2):245–270, 2006.

[OMG06] OMG. Mof core specification, v2.0, 2006. http://www.omg.org/docs/formal-/06-01-01.pdf.

[Ora00] Oracle8i"the xml enabled data management system. In *ICDE '00: Proceedings of the 16th International Conference on Data Engineering*, San Diego, California, USA, 2000. IEEE Computer Society.

[Ora08a] Oracle. Oracle bi publisher overview, 2008. http://www.oracle.com/technology/products/xml-publisher/index.html.

[Ora08b] Oracle. Oracle xml db, 2008. http://www.oracle.com/technology/tech/xml-/xmldb/index.html.

[Ora09] Oracle. Oracle berkeley db xml, 2009. http://www.oracle.com/database-/berkeley-db/xml/index.html.

[PAGM96] Yannis Papakonstantinou, Serge Abiteboul, and Hector Garcia-Molina. Object fusion in mediator systems. In *VLDB '96: Proceedings of the 22th International Conference on Very Large Data Bases*, pages 413–424, San Francisco, CA, USA, 1996. Morgan Kaufmann Publishers Inc.

[PD99] Norman W. Paton and Oscar Díaz. Active database systems. *ACM Comput. Surv.*, 31(1):63–103, 1999.

[PLO91] HweeHwa Pang, Hongjun Lu, and Beng Chin Ooi. An efficient semantic query optimization algorithm. In *Proceedings of the Seventh International Conference on Data Engineering*, pages 326–335, Washington, DC, USA, 1991. IEEE Computer Society.

[PPW03] G. Papamarkos, A. Poulovassilis, and P. Wood. Event-condition-action rule languages for the semantic web. In *In Proc. Workshop on Semantic Web and Databases, at VLDB'03*, Berlin, 2003.

[PV02] Yannis Papakonstantinou and Vasilis Vassalos. Architecture and implementation of an xquery-based information integration platform. *IEEE Data Eng. Bull.*, 25(1):18–26, 2002.

[RLS$^+$98] Johnathon Robie, Joe Lapp, David Schach, Michael Hyman, and Johnathon Marsh. Xml query language (xql), 1998. http://www.w3.org/TandS/QL/QL98/pp/xql.html.

[RS86] Lawrence A. Rowe and Kurt A. Shoens. Data abstraction, views and updates in rigel. pages 278–294, 1986.

[Rys01] Michael Rys. Bringing the internet to your database: Using sqlserver 2000 and xml to build loosely-coupled systems. In *Proceedings of the 17th International Conference on Data Engineering*, pages 465–472, Washington, DC, USA, 2001. IEEE Computer Society.

[Sch00] Uwe Schöning, editor. *Logik für Informatiker.* Spektrum Akademischer Verlag, 2000.

[Sch09] Schematron, 2009. http://www.schematron.com.

[SD95] Eric Simon and Angelika Kotz Dittrich. Promises and realities of active database systems. In *VLDB '95: Proceedings of the 21th International Conference on Very Large Data Bases*, pages 642–653, San Francisco, CA, USA, 1995. Morgan Kaufmann Publishers Inc.

[Shu00] Hua Shu. Using constraint satisfaction for view update. *J. Intell. Inf. Syst.*, 15(2):147–173, 2000.

[SJGP94] Michael Stonebraker, Anant Jhingran, Jeffrey Goh, and Spyros Potamianos. On rules, procedures, caching and views in database systems. pages 363–372, 1994.

[SNS05] Feng Shao, Antal Novak, and Jayavel Shanmugasundaram. Triggers over xml views of relational data. In *ICDE '05: Proceedings of the 21st International Conference on Data Engineering*, pages 483–484, Washington, DC, USA, 2005. IEEE Computer Society.

[SNS06] Feng Shao, Antal Novak, and Jayavel Shanmugasundaram. Triggers over nested views of relational data. *ACM Trans. Database Syst.*, 31(3):921–967, 2006.

[SO89] S. T. Shenoy and Z. M. Ozsoyoglu. Design and implementation of a semantic query optimizer. *IEEE Trans. on Knowl. and Data Eng.*, 1(3):344–361, 1989.

[SPD08] Mark Fussell Shankar Pal and Irwin Dolobowsky. Xml support in microsoft sql server 2005, 2008. http://msdn.microsoft.com/en-us/library/ms345117.aspx.

[SQL92] Information technology - database language sql, 1992. http://www.contrib.andrew.cmu.edu/ shadow/sql/sql1992.txt.

[Sun08a] Sun. Data access object, 2008. http://java.sun.com/blueprints/corej2eepatterns-/Patterns/DataAccessObject.html.

[Sun08b] Sun. J2ee connector architecture white paper - integrating java applications with existing enterprise applications, 2008. http://java.sun.com/javaee/overview/whitepapers/connector.jsp.

[Sun08c] Sun. Transfer object, 2008. http://java.sun.com/blueprints/patterns-/TransferObject.html.

[Sun08d] Sun. Transfer object assembler, 2008. http://java.sun.com/blueprints/corej2eepatterns-/Patterns/TransferObjectAssembler.html.

[Sun09] Sun. Java message service (jms), 2009. http://java.sun.com/products/jms/.

[TC97] Can Turker and Stefan Conrad. Towards maintaining integrity of federated databases. In *BIWIT '97: Proceedings of the 3rd Basque International Workshop on Information Technology (BIWIT '97)*, page 93, Washington, DC, USA, 1997. IEEE Computer Society.

[TFC83] Luiz Tucherman, Antonio L. Furtado, and Marco A. Casanova. A pragmatic approach to structured database design. In *VLDB '83: Proceedings of the 9th International Conference on Very Large Data Bases*, pages 219–231, San Francisco, CA, USA, 1983. Morgan Kaufmann Publishers Inc.

[TIHW01] Igor Tatarinov, Zachary G. Ives, Alon Y. Halevy, and Daniel S. Weld. Updating xml. In *SIGMOD '01: Proceedings of the 2001 ACM SIGMOD international conference on Management of data*, pages 413–424, New York, NY, USA, 2001. ACM.

[Ull97] Jeffrey D. Ullman. Information integration using logical views. In *ICDT '97: Proceedings of the 6th International Conference on Database Theory*, pages 19–40, London, UK, 1997. Springer-Verlag.

[VA96] Mark W. W. Vermeer and Peter M. G. Apers. The role of integrity constraints in database interoperation. In *VLDB '96: Proceedings of the 22th International Conference on Very Large Data Bases*, pages 425–435, San Francisco, CA, USA, 1996. Morgan Kaufmann Publishers Inc.

[VSCR00] Genoveva Vargas-Solar, Christine Collet, and Helena G. Ribeiro. Active services for federated databases. In *SAC '00: Proceedings of the 2000 ACM symposium on Applied computing*, pages 356–360, New York, NY, USA, 2000. ACM.

[W3C99a] W3C. Document type definition, 1999. http://www.w3.org/TR/REC-html40/sgml/dtd.html.

[W3C99b] W3C. Xsl transformations (xslt) 1.0, 1999. http://www.w3.org/TR/xslt.

[W3C06a] W3C. Extensible markup language (xml) 1.0 (fourth edition), 2006. http://www.w3.org/TR/2006/REC-xml-20060816/.

[W3C06b] W3C. Xml schema, 2006. http://www.w3.org/XML/Schema.

[W3C08a] W3C. Xquery update facility 1.0, 2008. http://www.w3.org/TR/xquery-update-10/.

[W3C08b] W3C. Xquery update facility 1.0 requirements, 2008. http://www.w3.org/TR/xquery-update-10-requirements/.

[Wan08] Xiaolong Wan. Enforcing constraints and triggers for active data services, 2008. Master Thesis, University of Stuttgart, Diplomarbeit Nr: 2714.

[Wid96] Jennifer Widom. The starburst active database rule system. *IEEE Transactions on Knowledge and Data Engineering*, 8(4), 1996.

[Wie92] Gio Wiederhold. Mediators in the architecture of future information systems. *Computer*, 25(3):38–49, 1992.

[WRM06] Ling Wang, Elke A. Rundensteiner, and Murali Mani. Updating xml views published over relational databases: towards the existence of a correct update mapping. *Data Knowl. Eng.*, 58(3):263–298, 2006.

[Xca08] Xcalia. Xcalia intermediation core, 2008. http://www.xcalia.com.

[xfr06] xfront. Xml schema: Best practices, 2006. http://www.xfront.com/BestPracticesHomepage.htm.

[XML00] XML:DB. Xupdate - xml update language., 2000. http://xmldb-org.sourceforge.net/xupdate/.