Universidade Federal de Pernambuco
Centro de Informática
Pós-Graduação em Ciências da Computação

Managing the Evolution of XML-based Mediation Queries

by

Bernadette Farias Lóscio

Tese de Doutorado

Recife, April 2003

Universidade Federal de Pernambuco
Centro de Informática
Pós-Graduação em Ciências da Computação

Managing the Evolution of XML-based Mediation Queries

Bernadette Farias Lóscio

Submitted in partial fullfilment of the requirements
for the degree of Doctor of Philosophy.

Supervisor: Profa. Dra. Ana Carolina Salgado

Recife, April 2003

To my mother Roceli, my father Roberto and
to my husband Luciano

Ackowledgments

This dissertation would not have become reality if it were not for the various people that directly and indirectly contributed to it. First of all, I would like to thank my advisor, Profa. Carol for an endless supply of suggestions and criticisms. I learned a lot from her not only how to do research, but also how to interact effectively with students. The example set by Profa. Carol had an important part in my decision to continue in academia.

I am deeply grateful to Professor Bouzeghoub who advised me on many technical topics related to my research during my PhD sandwich. I would also thank Professor Bouzeghoub for his warm hospitality. A special thanks for Zoubida Kedad for all interesting discussions we had during my stay in Laboratoire PRiSM. The ambiance between us was a great platform for churning out new ideas. I was lucky to meet some great people during my stay in Laboratoire PRiSM! Also, thanks to Assia for all the good times that we spent together.

Besides, I would also like to express my gratitude to Ceça, Guilherme, Luciano and Haroldo. They made substantial contributions to my work, especially in the implementation of some of the components of the prototype.

I would like to thanks to my family and my friends, who have given me support and encouragement over the past several years, for their patience during the times when I should have been paying attention to them, but instead I was absorbed in my studies. A special thanks to my parents Roberto and Roceli. I could not have come this far without their constant love, support and sacrifices. You are really special! Also, thanks to my sisters, Paula and Cláudia, and my brother Roberto, for their love and encouragement. Special thanks also to tia Fátima and tio Amaro for their love.

Most of all, I thank my husband, Luciano, for his constant love, patience and encouragement which made my PhD candidate life far less stressful. He helped me a lot throughout my graduate studies. He was always ready to say the magic words: “Don’t worry! Everything is going to be all right!”.

Thanks God!

Abstract

In recent years, the problem of integrating data from heterogeneous and autonomous data sources has received a great deal of attention from the database community research. This problem consists in providing a uniform view of these data sources (called mediation schema or global schema) and defining a set of queries which compute each object in the mediation schema (called mediation queries, mediation mappings or operational mappings).

Previous work in data integration can be classified according to the approach used to define the mediation mappings between the data sources and the global schema. The first approach is called global-as-view (GAV) and requires that each object of the global schema be expressed as a view (i.e. a query) on the data sources. In the other approach, called local-as-view (LAV), mediation mappings are defined in an opposite way; each object in a given source is defined as a view on the global schema.

In this work, we propose a mediator-based data integration system, which adopts the GAV approach. A distinguishing feature of this system is that besides integrating data it also deals with the problems concerning generation and maintenance of mediation queries. The proposed system uses XML as a common data model for data exchange and integration, and XML Schema language to represent the mediation schema and the source schemas. To provide a high-level abstraction for information described in an XML schema we propose a conceptual data model, called X-Entity model. We also present the process of converting an *XML Schema* to an X-Entity schema. This process is based on a set of rules that considers element declarations and type definitions of an *XML Schema* and generates the corresponding conceptual elements.

One of the main problems in the context of the GAV approach is the maintenance of the mappings between the mediation schema and the source schemas. In a dynamic environment, the mediation queries must be flexible enough in order to accommodate new data sources and new users' requirements. In this context, we address a novel and complex problem that consists in propagating a change event occurring at the source level or at the user level into the mediation level. To manage the evolution of the mediation level we have defined: i) a set of X-Entity schema change operations, ii) a set of propagation primitives reflecting the changes in the mediation level and iii) a set of propagation rules. We also propose a back-end process to execute the propagation of schema changes to the mediation queries.

We propose an incremental approach to develop the mediation schema and the mediation queries based on the evolution of the data source schemas and the evolution of the users' requirements. More precisely, if a new data source is added, for example, the mediation queries do not need to be recomputed from scratch. Instead, we can add the new data source to the existing queries. When an existing data source is removed, we check the queries, and the ones in which the deleted source appears may be removed or rewritten. In the same way, changes in the users' requirements can be reflected in the mediation schema and in the mediation queries for the cases where it is possible to do so. The proposed approach allows the mediation level to evolve incrementally and modifications can be handled easier increasing the system flexibility and scalability.

Resumo

Diversos sistemas de integração de dados têm sido propostos na literatura com o objetivo de prover acesso integrado a diferentes fontes de dados, que podem ser autônomas e heterogêneas. O problema de integração de dados consiste em oferecer uma visão uniforme das fontes de dados (chamada esquema de mediação ou esquema global) e definir um conjunto de consultas (chamadas consultas de mediação) as quais determinam como obter cada elemento do esquema de mediação em função dos dados armazenados nas fontes locais.

Sistemas de integração de dados podem ser classificados de acordo com a abordagem adotada para definição dos mapeamentos entre as fontes de dados e o esquema de mediação. Duas abordagens principais são apresentadas na literatura: *Visão Global* e *Visão Local*. Na abordagem *Visão Global* cada elemento do esquema de mediação é representado como uma visão sobre as fontes de dados. Na abordagem *Visão Local* cada elemento em uma dada fonte de dados é definido como uma visão sobre o esquema de mediação.

Uma das contribuições deste trabalho é a proposta de um sistema de integração de dados que adota a abordagem *Visão Global*. Um importante diferencial deste sistema é que além de prover acesso integrado a dados distribuídos e heterogêneos, o sistema também oferece soluções para os problemas relacionados à geração e à manutenção das consultas de mediação. Além disso, o sistema proposto usa XML como modelo de dados comum para troca e representação de dados. Para representar os esquemas das fontes de dados locais é adotada a linguagem XML Schema, proposta pelo W3C como linguagem padrão para definição de esquemas para classes de documentos XML. Para prover uma representação de mais alto nível para as informações descritas nos esquemas XML é proposto um modelo conceitual, chamado X-Entity. Além do modelo conceitual, também é apresentado o processo de conversão de um esquema XML (definido na linguagem XML Schema) para um esquema definido no modelo X-Entity.

O principal problema com o uso da abordagem *Visão Global* diz respeito à manutenção das consultas de mediação em conseqüência das atualizações nos esquemas das fontes de dados locais. Em ambientes dinâmicos, as consultas de mediação devem ser flexíveis a fim de permitir modificações nos esquemas locais, adição e remoção de fontes de dados e alterações nos requisitos de usuários. Para gerenciar a evolução do nível de mediação (esquema e consultas de mediação) foram desenvolvidos: i) um conjunto de operações que descrevem os diferentes tipos de evolução nas fontes locais e nos requisitos dos usuários, ii) um conjunto de primitivas de propagação que descrevem as modificações a serem realizadas no esquema e nas consultas de mediação e iii) um conjunto de regras de propagação. Este trabalho também propõe um processo de propagação que define como difundir os diferentes tipos de atualizações.

Este trabalho propõe uma abordagem incremental para o desenvolvimento do nível de mediação baseado na evolução dos esquemas das fontes locais e evolução dos requisitos dos usuários. Mais precisamente, a adição de uma nova fonte de dados ao sistema não implica que as consultas de mediação sejam completamente refeitas. Ao invés disso, é possível adicionar a nova fonte de dados às consultas existentes. Quando uma fonte de dados é removida, as consultas de mediação afetadas por esta remoção devem ser reescritas ou removidas. Da mesma forma, mudanças nos requisitos dos usuários também podem ser refletidas no nível de mediação. Esta solução permite a evolução incremental do nível de mediação aumentando tanto a flexibilidade quanto a escalabilidade do sistema de integração proposto.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

In recent years, the problem of integrating data from heterogeneous and autonomous data sources has received a great deal of attention from the database research community. This problem consists in providing a uniform view of these data sources (called mediation schema or global schema) and defining a set of queries which compute each object in the mediation schema (called mediation queries, mediation mappings or operational mappings). Various systems [Ambite et al. 1998, Baru et al. 1999, Bergamaschi et al. 1998, Chawathe et al. 1994, Cluet et al. 1998, Draper et al. 2001, Gardarin et al. 2002, Ives et al. 1999] have been proposed with the goal of integrating data from distributed data sources.

Previous work on data integration can be classified according to the approach used to define the mediation mappings between the data sources and the global schema [Halevy 2000, Levy 2000, Ullman 1997]. The first approach is called global-as-view (GAV) and requires that each object of the global schema should be expressed as a view (i.e. a query) on the data sources. Several projects, like Tsimmis [Chawathe et al. 1994], YAT [Cluet et al. 1998] and Disco [Tomasic 1998] adopt the GAV approach. In the other approach, called local-as-view (LAV), mediation mappings are defined in an opposite way; each object in a given source is defined as a view on the global schema. The LAV approach is used in the Information Manifold system [Kirk 1995], the OBSERVER system [Mena et al. 1996], the Infomaster system [Genesereth et al. 1997] and the PICSEL project [Goasdoué et al. 2000].

This work proposes a mediator-based data integration system [Lóscio et al. 2001], which uses the GAV approach. We adopted the GAV approach because it is more natural to implement and the user queries decomposition is a very simple task while in the LAV approach this process is very complicated and time-consuming. A distinguishing feature of this system is

that besides integrating data it also deals with the problems concerning generation and maintenance of mediation queries.

The definition of mediation queries is a difficult task in the context of large scalable systems, regarding the amount of metadata necessary to determine the queries. In our work, to help the designer during the process of mediation queries definition we use an adaptation of the approach for view expressions discovery proposed in [Kedad et al. 1999].

Another problem in the context of the GAV approach is the maintenance of the mappings between the mediation and the source schemas. In GAV approach, mediation queries are very sensitive to changings in the data sources and their adaptation may be a very complex task. In a dynamic environment, the mediation queries must be flexible enough in order to accommodate new data sources and new users' requirements. In this context, we address a novel and complex problem that consists in propagating a change event occurring at the source level or at the user level into the mediation level.

We propose an incremental approach to develop the mediation schema and the mediation queries [Lóscio et al. 2002b, Bouzeghoub et al. 2002] based on the evolution of the data source schemas and the evolution of the users' requirements. More precisely, if a new data source is added, for example, the mediation queries do not need to be recomputed from scratch. Instead, we can add the new data source to the existing queries. When an existing data source is removed, we check the queries, and the ones in which the deleted source appears may either be removed or rewritten. In the same way, changes in the users' requirements can be reflected in the mediation schema and in the mediation queries for the cases where it is possible to do so. The proposed approach allows the mediation level to evolve incrementally and modifications be handled easier, increasing the system flexibility and scalability.

As we know, one of the difficulties in integrating information from multiple data sources is their heterogeneous structure. To overcome this limitation, integration systems use a common data model for representing the sources' content and structure. As the data integration systems proposed in [Ambite et al. 1998, Baru et al. 1999, Draper et al. 2001, Gardarin et al. 2002, Ives et al. 1999], we use XML [Bray et al. 2000] as a common data model for data exchange and integration. We adopt XML due to its flexibility to represent both structured and semi-structured information. Moreover, to build a system with XML as its core, we have to provide a means of defining the structure and semantics of XML documents. In this work, we use XML Schema [Fallside 2001] to represent the mediation and the source schemas. We make a distinction between the terms XML schema and XML Schema. The first one refers to a general term designating a schema for a class of XML documents, while the second refers to the XML

Schema language. In the remainder of this work, we will consider only XML schemas defined in the XML Schema language, which will be denoted by *XML Schema*.

In this chapter, we present an overview of the focus of this thesis and some research issues related to our approach. The chapter concludes with a summary of the remainder of the thesis.

1.2 Thesis focus

The main contribution of this work is a solution for the problem of managing mediation queries evolution in the context of the proposed data integration system. However, to achieve this goal other problems had to be solved, including: i) how to deal with the hierarchical structure of the *XML Schemas* and ii) how to provide a high-level representation for mediation queries definitions. In this section, we give an overview of these problems by describing the research issues related to our work.

1.2.1 A conceptual model for XML schemas

We use the XML Schema language to represent the mediation and the exported schemas. These schemas will be used to validate the local data returned by the data sources as well as the integrated data returned by the mediator in response of a user query. Although being very useful for these tasks, an *XML Schema* is not suitable for tasks requiring knowledge about the semantics of the represented data. For example, during the mediation queries generation, it may be difficult to identify the elements which represent real world concepts and, therefore, should be associated with a mediation query.

To provide a high-level abstraction for information described in an XML schema we propose a conceptual data model, called X-Entity model. The main purpose of this model is to describe the component parts of an XML schema in a simple way by providing a means of better understanding the semantics of the represented data. The main concept of the X-Entity model is the entity type, which represents the structure of XML elements composed by other elements and attributes. In the X-Entity model, relationship types represent element-subelement relationships and references among elements.

The X-Entity model represents the hierarchical structure of XML schemas using a flat representation that puts in evidence entities and the relationships among them. Such representation provides a cleaner description for XML schemas by focusing only on semantically relevant concepts.

Another advantage of using the X-Entity model is that each entity can be seen and manipulated as an individual concept even if it belongs to a nested structured. Instead of having

nested entities, each entity has a set of relationships that represents its association with other entities.

Due to the hierarchical structure of XML schemas it would be hard to identify the scope in which a schema change should be propagated. However, using the X-Entity model each change will be defined with respect to a single entity of a data source schema and it will be propagated into one or more entities in the mediation schema.

1.2.2 Mediation queries generation

Finding rewritings for mediation queries becomes a more complex task when we try to directly perform the changes in the mediation query definition. Therefore, to facilitate the evolution process it is crucial to have a high-level representation for mediation queries. Such representation must be easy to maintain and it must be possible to develop an algorithm to automatically generate mediation queries definitions from this high-level abstraction.

To represent mediation queries, we use the formalism proposed by Kedad & Bouzeghoub [Kedad et al. 1999]. They propose an approach which provides a support to discover queries over a set of heterogeneous sources in a GAV context. Since they use the relational model as the common data model, their approach defines a solution space which identifies the set of potential queries corresponding to a given relation. The queries are represented through operation graphs, which describe the relevant relations and the operations to be applied among them in order to compute the integrated view. The operation graph describes all information that is relevant to compute a given integrated view. Another important issue to be considered is that an operation graph can be incrementally created and it can be easily modified.

We adapted that approach to our context, in such a way that, at the end of the mediation queries generation process, each mediation entity will be associated with a mediation query, which will be represented by an operation graph composed by a set of mapping entities and operations among them. Each mapping entity is derived from a source entity and it describes the attributes and relationships of the source entity that are relevant to compute the corresponding mediation entity.

This high-level representation facilitates the identification of the mediation entities that are affected by a data source schema change and that, consequently, must be rewritten. A mediation entity will be affected by an entity source change if one of its mapping entities will be affected by the data source change.

Moreover, it becomes easier to propagate data source changes into mediation queries. Considering that a mediation query consists of a set of operations applied to the mapping

entities, then the problem of mediation queries evolution consists, first in propagating these changes into the mapping entities, and secondly by rewriting the affected mediation queries in order to take into account the modifications in their set of mapping entities.

1.2.3 Mediation queries evolution

In this work, we address the problem of how to maintain the consistency of the mediation schema and the mediation queries when users' requirements or data sources' schemas change. We deal with this problem by considering a specific context of data integration, where a mediation schema (Figure 1.1) represents the reconciliation between users' requirements and the data sources' capabilities. The mediation schema describes an integrated view of data distributed in multiple data sources. Thus, users pose queries in terms of the mediation schema, rather than directly in terms of the source schemas. Each entity in the mediation schema is virtual, i.e. it is not actually stored anywhere, and it is computed from the integration of the distributed data.

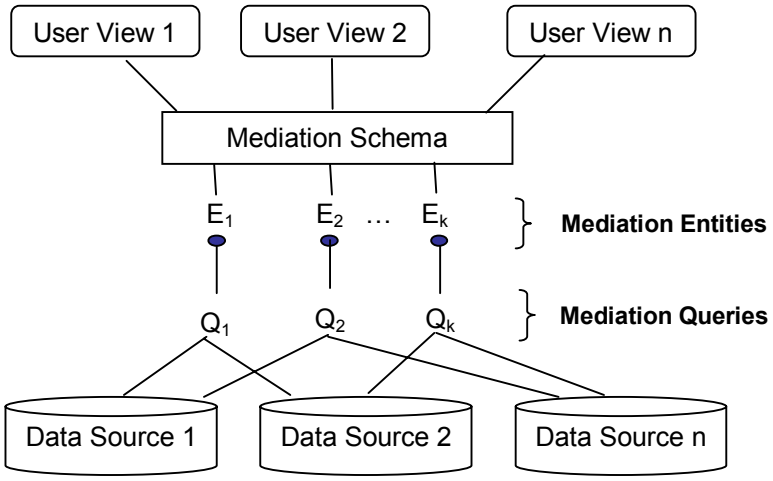

Figure 1.1 – Mediation schema definition

The mediation schema contains all the entities needed to answer the user queries, which can be computed from the data sources. Each entity E_i in the mediation schema is associated with a mediation query Q_i that computes the entity E_i over the set of data sources. Based on the mediation queries, previously defined, the system computes at run-time the answer for a user query by simply executing the necessary mediation queries and combining their results. In this approach, the sources that are relevant to compute a given entity in the mediation schema are determined in advance. This approach is less flexible in terms of dynamically selecting sources for answering queries. On the other hand, it scales well to a large number of sources since the search space for relevant data may become quite large as the number of sources increases.

One of the main challenges in data integration systems is to maintain the mediation schema consistent with the users' requirements evolution and to maintain the mediation queries consistent both with the mediation schema evolution and with source evolution, especially in the GAV approach where mediation queries are very sensitive to changes in source descriptions. To the best of our knowledge, only few aspects of this problem have been addressed so far.

In this context, two kinds of evolution have to be mainly dealt with: the evolution of the users' needs, and the evolution of the data sources:

i) *the evolution of the user needs*: it may consist of adding, removing or modifying an user requirement. These changes impact the mediation schema by adding, modifying or deleting an element from the mediation schema. Each change raised in the mediation schema may lead to the redefinition of some mediation queries. If this change can be reflected in these queries, the modification on the mediation schema is committed; otherwise the user is informed that the new requirement cannot be satisfied.

ii) *the evolution of the source schemas*: if a change occurs in a source schema, it has to be propagated to the mediation queries. The mediation queries are modified if the source elements on which they were defined are modified or when a source element is added or deleted. Some of the entities in the mediation schema may become no longer computable with respect to the changes raised at the source level.

1.3 Research issues

In the following, we summarize the main research issues concerning to our work.

- We propose a data integration system architecture that adopts the GAV approach and provides support for generation and maintenance of mediation queries. The system uses XML as the common data model to data exchange and integration. Moreover, the XML Schema language is used to represent both source schemas and the mediation schema. *XML Schemas* are used to validate the data exported by the data sources and the integrated data returned by the mediator.
- We also propose a conceptual model, called X-Entity, to describe the component parts of *XML Schemas* in a simpler way. X-Entity is devised to provide an effective support for the tasks of mediation queries generation and maintenance, by giving a conceptual representation of an *XML Schema* that puts in evidence classes of concepts and their relationships. We describe the process of converting an *XML Schema* to a conceptual schema defined in the X-Entity model. This process is based on a set of extraction rules that

considers element and type declarations of an *XML Schema* and generates the corresponding conceptual elements.

- We present an adaptation to the process of mediation queries generation proposed in [Kedad et al. 1999]. Such approach discovers view expressions in the context of the relational model. As we consider XML as our common data model we had to adapt it to the context of XML data. This approach provides a formalism, based on the concept of operation graphs, to represent mediation queries. One of the problems faced at the beginning of our work, concerning to the evolution of mediation queries, was the absence of a formalism to their representation. Using this approach, the problem of mediation queries evolution becomes the problem of maintaining the associated operation graphs.
- We propose a back-end process to manage the evolution of mediation queries according to the data source schemas changes and users' requirements changes. As the number of data sources increases we have to find the best way of synchronizing the propagation of events notified by different data sources and users in order to minimize the effort of generating new mediation queries. We present a set of X-Entity schema change operations representing the possible modifications in the data source schemas and in the users' requirements. We propose a set of propagation primitives describing the modifications at the mediation level and a set of propagation rules to determine the relevant propagation primitives to be applied after a schema change.
- We implemented a prototype of the proposed data integration system in collaboration with the Database Systems Research Group of Federal University of Pernambuco.

1.4 Plan of thesis

The remainder of this thesis is organized as follows:

- Chapter 2 gives an overview of the data integration research, describing the main approaches for data sources modeling and architectures for data integration. This chapter also discusses the main problems faced in data integration systems, including: heterogeneity of data sources, query reformulation, query optimization and view synchronization. As we propose an XML-based data integration system, we also discuss some aspects of the XML language. Finally, some related works are discussed.
- Chapter 3 provides an architectural overview of the proposed data integration system. This chapter describes the main components of the proposed system. It also introduces some of our key design decisions.

- Chapter 4 presents our approach to *XML Schemas* conceptual modeling. First, the X-Entity conceptual model is introduced and some basic features of the XML Schema language are discussed. Next, the process of converting an *XML Schema* to its corresponding X-Entity schema is described.
- Chapter 5 discusses our approach to mediation queries generation with a detailed description of the process of generating mediation queries for XML-based mediators.
- Chapter 6 presents the propagation of data source schema evolution and the propagation of users' requirements evolution to the mediation level. The schema changing operations, the propagation primitives, reflecting the schema changes at the mediation level, and a set of event-condition-action (ECA) rules specifying the propagation of schema changes are described. This chapter also introduces the propagation processes used in our approach to propagate data source schemas and users' requirements changes to the mediation level.
- Chapter 7 concludes the thesis with our research contributions and some future work.

Chapter 2

Data Integration

2.1 Introduction

The goal of a data integration system consists in offering a uniform interface to provide access to a collection of distributed data sources, which can be heterogeneous, autonomous and dynamic. The most important advantage of a data integration system is that it enables users to specify what they want, rather thinking about how to obtain the answers [Levy 1999].

In this chapter, we give an overview of the data integration research, describing the main approaches to data sources modeling and architectures for data integration. We also discuss the main problems faced in data integration systems, including: heterogeneity of data sources, query reformulation, query optimization and view synchronization. As we propose an XML-based data integration system, we also discuss few aspects of the XML language, which has been used in several data integration systems as the commom model for data exchange and integration. We conclude with some related work.

2.2 Approaches to data sources modeling

An important decision in building a data integration system concerns the specification of the mappings between the mediation schema and the data sources. Several researches have been developed in this area and two basic approaches have been used to specify the mappings between the sources and the global schema: Global as View (GAV) and Local as View (LAV) [Halevy 2000, Levy 2000, Ullman 1997].

Global as View. The GAV approach requires that the global schema should be expressed in terms of the data sources, i.e., every element in the mediation schema is associated with a view over the data sources. This view specifies how to get the data of the mediation schema by queries over the data sources. This approach is more natural to implement and the query

processing is very simple. As the elements in the mediation schema are defined in terms of the sources, to answer a query submitted to the system it is just necessary to unfold the definitions of mediation schema elements in order to obtain the corresponding data.

The main disadvantage of this approach concerns to the maintenance of the mapping views. This task can be very complex and time-consuming. For example, the addition of a new data source may require changing the views that define the concepts in the mediation schema. If the new data source contains relevant information to some elements of the mediation schema, then their corresponding definitions must be modified. The modification or the removal of an existing data source may also require a change in the mediation schema.

Local as View. The LAV approach requires that the mediation schema should be specified independently from the sources. In this approach the information content of each data source is specified in terms of a view over the mediation schema. Data integration systems that use the LAV approach are more extensible and the addition of new data sources can be easier handled. Adding a new data source to the system requires only providing the definition of the source, and does not, necessarily, involve changes in the mediation schema. Existing data sources can also be modified without changing the mediation schema. The main disadvantage of the LAV approach is that the query reformulation is a very complex and time-consuming task. Different from the GAV approach, query answering in LAV is not based on a simple unfolding strategy. Since the data sources are defined as views over the global schema, the problem of query reformulation consists in how to answer queries defined over the global schema using only the views describing the data sources. The problem of query reformulation is similar to the problem of answering queries using views [Levy et al. 1995].

2.3 Approaches for data integration

Another important decision in building a data integration system is whether to take a materialized or a virtual approach to data integration. In the following, we describe both approaches in more details.

Virtual approach. In this approach, the data remains in the data sources and the user queries posed to the system are decomposed at run time into sub-queries on the data sources. The results of these sub-queries must be integrated in order to obtain the answer for the corresponding user query. One of the main advantages of this approach is that the data is guaranteed to be fresh at query time. However, this approach is disadvantageous with respect to the possibility of existing unavailable data sources and the query response time is often very high mainly, because a large number of data sources may be accessed to answer most queries.

The mediator architecture [Wiederhold 1992, Abiteboul et al. 2000], presented in Figure 2.1, adopts the virtual approach to data integration. The wrappers, in this architecture, provide access to heterogeneous data sources by converting application queries into source specific queries and it converts the data returned by the source into the common model. A mediator is a facility that supports an integrated view over multiple data sources. A schema for the integrated view is available from the mediator, and user queries can be made against that schema. The mediator receives queries posed to the system and decomposes them into queries to be executed in the remote data sources. When the mediator receives the results from the corresponding data sources then it integrates the data and it returns the integrated data to the user.

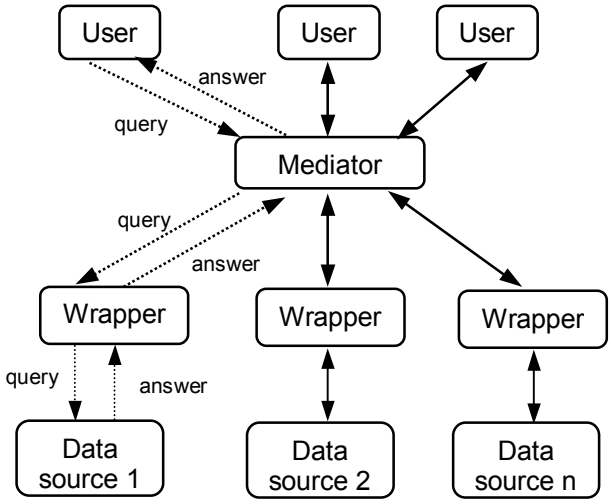

Figure 2.1 – Mediator architecture

Materialized approach. In this approach data are previously accessed, cleaned, integrated and stored in a data repository and the queries submitted to the integration system are evaluated in this repository without direct access to the data sources. This approach is better suited to cope with applications where the users need high performance at query time and they do not require fresh data. Figure 2.2 presents the data warehouse architecture, which adopts the materialized approach to data integration.

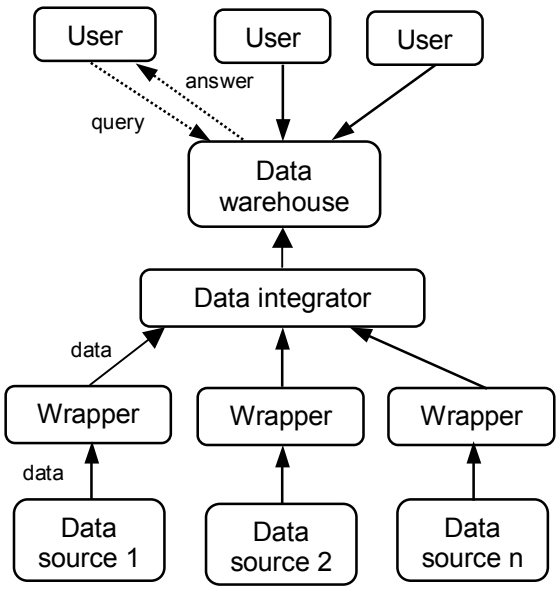

Figure 2.2 – Data warehouse architecture

One of the main problems to be considered in this architecture is the maintenance of the materialized data, which is the process of updating the data repository in response to changes to the underlying data [Gupta et al. 1995a, Widom 1995]. The problem of keeping the integrated data synchronized with the remote data sources is the same as the problem of maintaining materialized views in a distributed environment.

Basically, there are two strategies for view maintenance. The first one consists in recomputing all the integrated data from scratch. The second, called incremental maintenance, consists in modifying the integrated data only in response to changes on the underlying data [Abiteboul et al. 1998, Zhou et al. 1996]. The incremental maintenance is more efficient than re-computation, but it can be very expensive, since it may require querying the remote data sources. The system may have to issue queries to some of the data sources in order to obtain some additional data to correctly update the integrated views. A view is called self-maintainable when no additional queries over remote data sources are required to maintain the view [Tompa et al. 1988, Huyn 1997].

Among the other traditional architectures for data integration, we can mention the federated database architecture [Sheth et al. 1990] and the multidatabase [Elmagarmid et al. 1999].

– *Federated database*: a federated database system [Kim 1995, Sheth 1990] is a collection of autonomous database systems which participate in a federation to share data. A key issue in a federation is the cooperation among independent systems. Besides of a global schema, the federation offers multiple federated schemas in function of the application requirements using the federation. In general, federated schemas are defined manually.

There are two classical categories of federated systems: strongly coupled and loosely coupled. In the first, there is an admnistrator who is responsible for creating the federated schemas and controling the access to the database components. A federated system is considered loosely coupled when the user is responsible for creating and controling the federation.

– *Multidatabase*: in the multidatabase architecture [Elmagarmid 1999], there is no global schema. It offers a multidatabase language which supports queries defined over multiple databases. In this architecture, the participating databases have more autonomy. However, there is no notion of integrated schema and users must know the schemas of the component databases and have to solve the heterogeneity conflicts that may arise during query execution.

2.4 Main problems with data integration

In this section, we discuss some basic problems that are faced in building a data integration system. Some of these problems as heterogeneity and data sources modeling have received a great attention from the database research community. On the other hand, the view synchronization problem has not been broadly discussed. In this work, we focus on the view synchronization problem.

Heterogeneity. A basic problem in data integration systems is the heterogeneity, which arises in many forms, ranging from the hardware and software platform that a data source is based on, to the data model and schema used to provide logical structure for the stored data, to the very kinds of data that are being stored. The semantic heterogeneity, for example, is the result of representing the same information or overlapping data in different ways [Hull 1997]. In general, the data sources already exist and they are planned using different data models and schemas. A data integration system must overcome this heterogeneity to offer an integrated view of the data distributed in distinct data sources.

Data sources modeling and query reformulation. One of the problems in a data integration system, which adopts the virtual approach, is the query reformulation. As mentioned earlier, user queries posed to the system must be decomposed into queries to be evaluated on the remote data sources. Defining queries over tens or hundreds of heterogeneous data sources can be a very complex task. To reformulate a user query on the mediation schema into a query that refers directly to the source schemas, the system requires a complete and perfect understanding of the semantics of these sources. A description of an information source must specify these contents, constraints on the contents of the source, completeness and reliability, and finally the

query processing capabilities of the source [Florescu et al. 1998]. Besides of guaranteeing that the reformulation is semantically correct (i.e. the answers obtained from the sources will actually be correct answers to the query), it is also important to guarantee that irrelevant data sources have not been accessed, in order to have a better performance in the user query execution. As mentioned earlier, the complexity of the process of query reformulation depends on the approach adopted to define the mappings between the mediation schema and the source schemas.

View synchronization. In a data integration system, the data sources are autonomous and dynamic, updating not only their contents but also their schemas, and joining or leaving the system frequently. Therefore, the data integration system must evolve in order to become consistent with the local data sources. Most of the approaches used in data integration systems are based on previously defined static views which gather information from a fixed set of heterogeneous data sources and a fixed set of user requirements, and provide the user with a uniform view of the distributed information. In this context, a new problem, called view synchronization, must be considered. The view synchronization corresponds to the process of view definition adaptation triggered by changes of remote data sources. The problem of view synchronization was defined in [Nica et al. 1999] and until now has received little attention in the literature. The work proposed in [Nica 1999] presents a taxonomy of view adaptation problems in evolving environments based upon types of changes that either a view or a data source can undergo. Any process that changes the view definition or the view extent is referred as a view adaptation problem.

2.5 The XML model

XML (*Extensible Markup Language*) is the new standard format for data representation and exchange over the Web, currently being standardized by the World Wide Web Consortium [Bray et al. 2000]. XML is very flexible and provides a mean of representing both structured and semi-structured data [Buneman 1997]. Elements are the main building blocks of XML documents. An element may contain other elements, called child elements, or may contain character data, optionally mixed with child elements. An element is bounded by matching starting and ending tags such as <author> and </author>. Besides, an element may have a set of attribute specifications.

Because of the flexibility of XML to represent both structured and semi-structured data and the trend of using XML as the standard format for data representation and exchange over the Web, several data integration systems use XML as their common data model [Baru et al. 1999,

Draper et al. 2001, Gardarin et al. 2002, Ives et al. 1999]. These systems also adopt XML related standards, such as XML schema languages and XML query languages. In the following sections, we present some of the XML related standards proposed in the literature.

2.5.1 XML schema languages

An XML schema is a common vocabulary for applications exchanging data, which describes types of elements that can participate in a given class of XML documents. Using an XML schema it is possible to specify several constraints on an XML document, for example whether the elements and attributes are either required or optional and which element and attribute types are allowed. In the following, we briefly review some XML schema languages found in the literature. [Lee et al. 2000] provides a comparative analysis of these languages.

- *DTD (Document Type Definition)*: it was the first XML schema language proposed in the literature. It has very limited capabilities compared to other XML schema languages. Despite its limitations, it has currently been the most used XML schema language.
- *XML Schema* [Fallside 2001, Biron et al. 2001, Thompson et al. 2000]: it is the standard XML schema language proposed by the World Wide Web Consortium (W3C). It is more expressive than DTD and has many additional resources for XML schemas definition, including: a great variety of primitive data types, user-defined data types and inheritance for attributes and elements.
- *SOX (Schema for Object-Oriented XML)*[Davidson et al. 1999]: it was initially proposed to support the development of large-scale, distributed electronic commerce applications but it is also applicable across the whole range of markup applications. SOX is an alternative schema language and can be used to define the basic class of document types (with the exception of external parsed entities). However, SOX extends the language of DTDs definitions by supporting: an extensive (and extensible) set of datatypes, inheritance among element types, namespaces, polymorphic content, embedded documentation and features to enable robust distributed schema management.
- *Schematron* [Jellife et al. 2000]: it differs from the other XML schema languages because it is not based on the definition of grammars but allows the validation of schemas by using patterns. It is based on a simple action: first, find some context nodes in the document (typically an element) based on the XPath path criteria and then, check to see if some other XPath expressions are true for each of these nodes.

– *DSD (Document Structure Description)* [Klarlund et al. 2000]: it was produced as a result of collaborative research between AT&T and BRICS University in Denmark. DSD provides a context-dependent description of elements and attributes, flexible default insertion mechanisms and expressive power close to XSLT [Clark 1999b].

– *Relax (Regular Language description for XML)* [Makoto 2000]: it is both simple and built on a solid mathematical foundation. It was first published in March 2000 as a Japanese ISO Standard Technical Report written by Murata Makoto and it was later approved as an ISO/IEC Technical Report. RELAX NG (RELAX New Generation) is the result of a merger of RELAX and TREX [Clark 2000]. RELAX NG is now an official specification of the OASIS RELAX NG Technical Committee and will probably progress to become an ISO/IEC TR.

In this work, we adopt XML Schema to represent both the mediation schema and exported schemas. As XML Schema is the standard XML schema language recommended by the W3C, it will probably become the standard format to publish schemas for XML data.

2.5.2 XML query languages

Since XML data is very different from conventional data, traditional query languages can not be used to query XML data. Therefore, several XML query languages have been proposed in the literature [Ceri et al. 1999, Chamberlin et al. 2001, Clark 1999b, Deutsch 1999, Robie98a, Robie 1998b]. In [Bonifati et al. 2000], is presented a comparison of some query languages for XML, focusing on their common features and differences. In the following, we briefly review some of these languages.

– *XML-QL* [Deutsch 1999]: it was proposed by AT&T Labs in the context of the Strudel project. This language extends the SQL language with an implicit clause CONSTRUCT which permits the construction of XML documents as the result of a query. This language proposes the use of patterns to find specific data in a given XML document. It also provides means to transform XML data in order to facilitate the integration of XML data from different data sources.

– *XML-GL* [Ceri et al. 1999]: it was developed in the Politecnico di Milano to be a graphical query language. It is suitable for supporting a user-friendly interface because all elements are visually displayed. XML documents and DTDs are represented as labelled graphs.

– *XSL (Extensible Stylesheet Language)* [Clark 1999b, Schach et al. 1998]: it was designed by the W3C XSL working group. A XSL document consists of a collection of template rules;

each rule is composed by two components: a pattern and a template. The pattern is matched against nodes in the source document and the template is instantiated to compose the result tree.

– *XQL* [Robie98a, Robie 1998b] it can be considered a natural extension of the XSL language. It was designed to be sintatically simpler and compact, but with reduced expressive power. It was designed by J. Robie (Texcel Inc.), J. Lapp (webMethods, Inc.) and D. Schach (Microsoft Corporation).

– *XQuery* [Chamberlin et al. 2001]: it is the standard XML query language proposed by the W3C. The origin of XQuery is another XML query language called Quilt [Chamberlin00], which inherited diverse characteristics of other languages, such as: XPath [Clark 1999a], XQL [Robie 1998a], XML-QL [Deutsch et al. 1999], Lorel [Abiteboul et al. 1997] and YATL [Cluet et al. 1998].

2.5.3 The XML data models

Because XML documents are tree-structured, the XML data models are defined using conventional terminology for trees. To illustrate the concepts of an XML data model, we briefly describe the XQuery 1.0 and XPath 2.0 Data Model [Fernandez et al. 2001], which is the data model for XQuery 1.0. Such XML data model is a node-labeled, tree-shaped graph, where the nodes represent information about the document, element, attribute, text, namespace, processing instruction, and comment. A tree contains a root plus all nodes that are directly or indirectly reachable from the root. Every node belongs to exactly one tree, and every tree has exactly one root node. The data model associates type information with element nodes, attribute nodes and atomic values. Every value handled by the data model is a sequence of zero or more items: an item is either a node or an atomic value and a sequence is an ordered collection of nodes, atomic values, or any mixture of nodes and atomic values.

2.5.4 Algebras for XML

As XML data is semi-structured, the algebras developed for relational or object-oriented data cannot be directly used for XML queries. Most XML Algebras have operators which operate on a model which is similar to the XML Query data model [Fernandez et al. 2001]. However, the algebras are quite different from each other. In [Chinwala et al. 2001] a comparison of the following XML algebras is presented: IBM Algebra [Beech et al. 1999], Niagara Algebra [Galanis et al. 2001], YATL Algebra [Christophides et al. 2000], Lore [McHugh et al. 1999] and AT&T Algebra [Fernandez et al. 2001]. In the following, we present some characteristics

of these algebras along with the TAX algebra [Jagadish et al. 2001] and the XAT algebra [Zhang et al. 2002] that were not considered in that comparison.

- *Lore* [McHugh et al. 1999] and *YATL algebra* [Christophides et al. 2000]: while the IBM, AT&T and Niagara algebras were proposed as stand alone XML query algebras, Lore and YATL algebras were developed for the Lore database system and YAT data integration system, respectively. Thus, the algebra operators proposed by Lore could be used to generate query plans that could be efficiently executed using the physical index operators of the Lore database system. Similarly, since YATL was developed for an XML based data integration system, the optimization strategies are focused towards efficiently querying distributed data.

- *IBM* [Beech et al. 1999] and *Niagara algebra* [Galanis et al. 2001]: the IBM algebra, though not being a specific system like Lore, suffers from some drawbacks such as complex query structures with a large number of variable bindings and a lack of optimization rules. The Niagara algebra, which is the most recently proposed algebra, has operators which are similar to the IBM algebra which in turn has similar operators to object algebra operators. Niagara was developed to overcome the drawbacks of the IBM algebra.

- *AT&T algebra* [Fernandez et al. 2001]: the AT&T algebra was selected by the W3C as the proposed standard XML algebra [Fankhauser et al. 2001]. In a July 2001 revision [Draper et al. 2002] to the original algebra document by W3C, there have been some syntactical changes and now the document is titled as the proposed semantics of XQuery.

- *TAX algebra (A Tree algebra for XML)*[Jagadish et al. 2001]: the TAX algebra was defined in the context of the TIMBER XML database system [Jagadish et al. 2002]. TAX is used at its core for query evaluation and optmization. TAX extends the relational algebra by considering collections of ordered labeled trees instead of relations as the basic unit of manipulation. In spite of the potentially complex structure of trees involved, and the heterogeneity in a collection, TAX has just a couple of operators in addition to the relational algebra.

- *XAT algebra (XML Algebra for the Rainbow System)*[Zhang et al. 2002b]: this algebra was defined in the context of the Rainbow system [Zhang et al. 2002a], which is used to manage and query XML data stored in heterogeneous systems, in particular, in the relational format. The core part of the Rainbow system is the XAT algebra, which is the

foundation for the query engine to access both XML data and relational data. The XAT data model is an order-sensitive table called XAT table, which is an extended relational table with XML domains and supporting collections. The algebra supports operators: i) XML operators, which are used to represent the XML document related operations, ii) SQL operators, which correspond to the relational complete subset of the XAT algebra and iii)special operators include the operators temporarily used in different phases of optmization and the operators shared both by the class of XML operators and of SQL operators.

2.6 Related work

In this work, we propose a mediator-based data integration system that offers an integrated view of data distributed in several autonomous and heterogeneous data sources [Lóscio et al. 2001, Lóscio et al. 2002a]. One distinguishing feature of the proposed system is that it presents solutions for the problems concerning mediation queries generation and maintenance in dynamic environments. In this section, we present some data integration systems focusing their approach for modeling data sources (Global as View or Local as View) and their key design decisions concerning mediation queries generation. Next, we give an overview of some work directly related to the problem of schema evolution, which is our main focus. We conclude by discussing some conceptual models for XML schemas.

2.6.1 Data integration systems

Initially, data integration systems were developed to provide integrated access to distributed databases with well-defined structures. Later, with the increase of information available in the Web and the growth of its popularity, new data integration systems were developed with the goal of integrating both structured and semi-structured data sources, including: TSIMMIS [Chawathe et al. 1994], MOMIS [Bergamaschi et al. 1998], Ariadne [Ambite et al. 1998], MIX [Baru et al. 1999], Tukwila [Ives et al. 1999], Nimble Integration Suite [Draper et al. 2001] and e-XML Data Integration Suite [Gardarin et al. 2002].

In [Chawathe et al. 1994], the TSIMMIS (The Stanford-IBM Manager of Multiple Information Sources) is proposed, a mediator-based system that besides supplying data integration mechanisms also offers tools to facilitate the wrapper and mediator generation. A rule-based language, called Mediator Specification Language (MSL), is adopted in TSIMMIS to allow the declarative specification of mediation queries, which describe the elements of the mediation schema in terms of the data sources. The mediation queries are defined manually and

they are used to automatically or semi-automatically generate the mediator. TSIMMIS adopts a self-describing object-oriented data model, called OEM, which allows the representation of structured and semi-structured data. End users can access information either by writing applications that request OEM objects or by using one of the developed browsing tools. The end-user query language adopted in TSIMMIS is LOREL [Abiteboul et al. 1997], an OQL (Object Query Language) based query language for the OEM model.

MOMIS (Mediator envirOnment for Multiple Information Sources) [Bergamaschi et al. 1998] is a framework for integration of structured and semi-structured data that offers an integrated view of all the participating data sources, called Global Virtual View. MOMIS adopts ODM_{I3} to represent integrated information and to describe the data sources it adopts the ODL_{I3} language. ODM_{I3} and ODL_{I3} are subsets of the corresponding ones in ODMG (Object Database Management Group) [Cattell et. al 2000]. For each global class belonging to the Global Virtual View it is generated a mapping-table storing all the intensional mappings between the global class and the local classes. MOMIS offers a set of techniques to help the designer to face problems that arise when integrating pre-existing information sources. One of the key issues in the MOMIS system is a Common Thesaurus composed by intentional and extensional relationships, describing the correspondences among classes and attributes of source schemas.

In [Arens et al. 1993], is presented a data integration system, called SIMS, which was considered initially for integration of data stored in different databases, and later it was customized for the Web. This adaptation originated a system for integration of semi-structured web data sources, called ARIADNE [Ambite et al. 1998]. To build an application in SIMS, a user creates a domain model using the Loom knowledge representation language and describes the source contents in terms of this model. The domain model describes object classes, their attributes, and the relationships among them. SIMS accepts queries in this domain-level language, process these queries, and returns the requested data. To minimize the cost of query processing SIMS, adopts an hybrid approach to data sources modeling. First the sources are defined in terms of the global domain model and then compiled into axioms that define the global model in terms of the sources. An integration axiom specifies a particular way in which available sources can be combined to provide the data for a class belonging to the global model. As presented in [Ambite et al. 2001], the integration axioms may be automatically generated.

[Baru et al. 1999] presents a wrapper-mediator system, named MIX (Mediation of Information using XML), which employs XML for data modeling and interchange between heterogeneous data sources. MIX adopts the GAV approach to define the mediator views, which are provided by the mediation engineer and expressed in XMAS (XML Matching and

Structuring Language), a declarative XML query language. Queries in MIX are also expressed in XMAS and generated by a graphical user interface. To facilitate query formulation and for optimization purposes, MIX employs XML DTDs as a structural description of the exchanged data. Similar to the other data integration systems mentioned earlier, MIX adopts the virtual approach to query evaluation.

The Tukwila [Ives et al. 1999] data integration system is designed to scale up to the amounts of data transmissible across intranets and the Internet, with large numbers of data sources. The key issue of the Tukwila is that it has an adaptive query execution system. When the system receives a query then it intelligently processes the query, reading data across the network and responding to data source sizes, network conditions, and other factors. A highly efficient query reformulation algorithm, MiniCon, maps the input query from the mediation schema to the data sources [Pottinger et al. 2000]. Tukwila adopts a mediation schema to represent a particular application domain and data sources are mapped as views over the mediation schema. It also provides integrated support for efficient processing of XML data, based on the x-scan operator, which efficiently processes virtual XML data as it reaches the system. The latest versions of Tukwila are built around an adaptive query processing architecture for XML, and can perfectly combine XML and relational data into new XML content.

The Nimble Integration Suite [Draper et al. 2001] is an XML-based information integration software platform. Users and applications interact with the Nimble system using a set of mediation schemas, which are definitions of views over the data source schemas. It has a metadata server containing the mappings between the mediation schema and the sources which is used to decompose XML-QL queries into data source queries. One of the distinguishing features of Nimble is that it behaves intelligently when some data sources are not available by providing partial results and indicating to the users that results are not complete. It adopts an hybrid approach where it is possible to specify which data sources should be materialized in a local store and which should be refreshed on demand.

The e-XML Data Integration Suite [Gardarin et al. 2002] also adopts XML as the integration model. Besides XML, it also uses some XML related standards as XQuery, XML Schema, Dom, SAX, SOAP, Xforms and XSL. The e-XML suite is a collection of components built to work and cooperate with relational existing DBMSs (Oracle, DB2, SQL Server, TimesTen and Postgres).

Table 2.1 presents a comparison of the mentioned data integration systems. As we may observe, among the presented systems, most of them use the GAV approach to data sources modeling. As mentioned before, the GAV approach is easier to implement and the user query

decomposition process is more efficient. Since most of them adopt the virtual approach to data integration, we may conclude that the presented systems are more concerned in providing fresh data than in providing high performance at query time. Due to the flexibility of XML to represent both structured and semi-structured information, and to the ease with which one can convert any data to XML, there is an increasing interest in using it as a common data model for data integration. The use of XML as the common data model by the most recent proposals, demonstrates that XML is becoming the standard format for data exchanging. As presented in Table 2.1, one relevant point which is not discussed in details concerns the generation and specification of the mappings between the mediation schema and the source schemas. Such tasks are very costly and time consuming, therefore it is crucial to have algorithms to generate them. Among the presented systems, just MOMIS and SIMS use algorithms to generate the mediation queries. In TSIMMIS and MIX the mediation queries are executed manually. The data integration systems which adopt XML as the common data model do not present details about the mediation queries generation. As the structure of XML data is more flexible than the structure of conventional data, then the process of generating mediation queries is more complex. In the relational model, for example, a relation is composed by a set of tuples having the same schema and consisting of attributes, which are atomics and monovalued. On the other hand, XML elements may be composed by other elements and attributes, and elements with the same type may have different structures.

Table 2.1 – Comparison of data integration systems

	GAV x LAV	Virtual x Materalized	Integration model	Query language	Mappings specification	Mappings Generation
TSIMMIS	GAV	Virtual	OEM	Lorel	MSL	Provided by the mediation engineer
MOMIS	GAV	Virtual	ODM_{I3}/ODL_{I3}	OQL_{I3}	Mapping tables	Generated by algorithm
SIMS	Hibrid	Virtual	LOOM	--	LOOM	Generated by algorithm
MIX	GAV	Virtual	XML	XMAS	XMAS	Provided by the mediation engineer
e-XML		Hibrid	XML	XQuery	--	--
NIMBLE	LAV	Hibrid	XML	XQuery	--	--
TUKWILA	LAV	Virtual	XML	--	--	--

2.6.2 Schema evolution in data integration systems

One of the main problems faced when maintaining long-lived application systems is to handle database schema changes. These changes may occur not only because it is difficult to

completely determine the final database schema for many complex applications but also because the users' requirements change frequently. As suggested in some works [Marche 1993, Sjoberg 1993], handling schema changes is an inevitable task not only during the development of a system but also once a system has become operational.

Traditionally, the goal of schema evolution approaches is to allow the evolution of database schemas while changing the data to conform them to the modified schema. Schema evolution has been broadly discussed in the context of object-oriented database systems. Studies on this area have focused on (i) defining sets of schema evolution operations [Banerjee et al. 1987, Lerner 1996, Claypool et al. 1998], (ii) providing mechanisms to make data and the system itself more available during the schema evolution process [Ferrandina et al. 1994a, Ferrandina et al. 1994b] and (iii) providing support for existing applications that depend on the old schema, when other applications change the shared schema according to their own requirements [Lautemann 1997, Ra et al. 1997, Rundensteiner et al. 1998].

Another important issue is providing support for schema evolution in the context of data integration systems. As we know, local data sources are often autonomous and may change in their both structures and concepts. New sources may be added to the system or some sources may be removed from the system either because of their irrelevance or because of their unavailability.

A lot of work have studied the problem of materialized view maintenance [Gupta et al. 1995a], which consists in updating a materialized view in response to changes to the underlying data. It is important to observe that this is not the focus of our work. We are interested in providing mechanisms to correctly update the mappings between the mediation schema and the distributed sources after data source schemas changes and users' requirements changes. Few researches have discussed some aspects related to this problem [Ambite et al. 2001, McBrien et al. 2002, Nica et al. 1999].

The algorithm to discover integration axioms presented in [Ambite et al. 2001] is incremental, which means that when new sources are added, the system can efficiently update the axioms, but no details on how this could be achieved nor examples are given. Besides, in case of deleting a source the algorithm must start from scratch.

The problem of schema evolution is also discussed in [McBrien et al. 2002], which provides an approach to handle both schema integration and schema evolution in heterogeneous database architectures. The proposed approach is based on a framework for schema transformation, which consists of a hypergraph-based common data model and a set of primitive schema transformations defined for this model. Source schemas are integrated into a

global schema by applying a sequence of primitive transformations to them. This set of transformations can also be used to systematically adapt the global schema and the global query translation pathways after changes to the source schemas.

The work presented in [Nica et al. 1999] also investigates the view evolution problem in information integration systems. The authors propose the Evolvable View Environment (EVE) framework as a generic approach to solve issues related to view evolution under schema changes for both view definition adaptation and view extent maintenance after synchronization. EVE uses materialized views for data integration. They propose some synchronization algorithms to evolve a view definition by finding appropriate replacements for affected view components based on available meta-knowledge, and by dropping non-essential view components. Unlike the strategies proposed for query rewriting using views [Levy et al. 1996, Srivastava et al. 1996], the proposed algorithms find view rewritings that are not necessarily equivalent to the original definition. They use the relational data model as the common data model and they also propose an extended view definition language (derived from SQL), which allows users to explicitly define evolution preferences for changing the semantics of the view. In this way, it is possible to accept view rewritings that preserve only indispensable attributes if preserving all is not possible.

2.6.3 Conceptual modeling of XML schemas

Some recent works have discussed the conceptual data modeling of XML schemas using the ER model. In [Psaila 2000] is proposed the ERX conceptual model, an evolution of the classical ER model which provides specific features that are suitable to model large collections of XML documents. More precisely, ERX extends the ER model to allow the representation of style sheets and a collection of documents conforming to a DTD data. ERX is devised to be effective for building advanced XML processors that have to manipulate complex XML documents or multiple classes of documents at the same time.

The work presented in [Mani et al. 2001] proposes a new notation, called XGrammar, to formalize the most important features from the proposed XML schema languages. Mani et. al. also extends the ER model with additional features (order in a binary relationship and element-subelement relationship) to support the XML model. A conversion between XGrammar and the extended ER model is also discussed.

In [Passi et al. 2002], is proposed an object-oriented data model, called XSDM (XML Schema Data Model), to represent *XML Schemas*. XSDM was proposed as a model for XML Schema integration, i.e., during the first step of the schema integration *XML Schemas* are

translated into the XSDM notation. A briefly discussion of the process of converting an *XML Schema* to the XSDM notation is presented.

In [Mello et al. 2001], is described a semi-automatic process for converting a DTD to a conceptual schema in a canonical conceptual model, which is a mix of the ORM/NIAM [Halphin 1998] and EER models. As proposed in [Passi et al. 2002] the main focus of this work is the schema integration. A broadly discussion of the conversion from a DTD to its corresponding conceptual schema is presented.

Other related work include a mapping from XML Schema to an extended UML [Booch et al. 1999], a mapping from ORM to XML Schema and a mapping between UML class diagrams and *XML Schemas* [Bird et al. 2000]. Also other work discuss the conversion of *XML Schemas* (or DTDs) to relations for storage [Bohannon et al. 2002, Florescu et al. 1999].

2.7 Concluding remarks

In this chapter, we gave an overview of the data integration research area. First, we introduced the main approaches for data sources modeling: GAV and LAV. In the GAV approach query processing is easier than in the LAV, however the LAV approach ensures an easier extensibility of the data integration system. We also reviewed the main architectures for data integration, including: mediators and data warehouse. The main difference between them is that the first one offers a virtual integrated view of the distributed data, while the second offers a materialized integrated view.

We also discussed the main problems faced in data integration systems, including: heterogeneity of data sources, query reformulation and view synchronization. Since we propose an XML-based data integration system, we then discussed some XML schema languages, XML query languages, XML algebras and XML data models.

We also reviewed some data integration systems focusing their approach for modeling data sources and their key design decisions. These systems have as common goal offering an integrated view of data distributed in heterogeneous data sources. However, to achieve this goal they use different architectures and, generally, they use distinct data models and query languages. Except for the most recent proposals, which adopt XML and XQuery as the basis for the data integration process.

We also presented some work, which more directly discuss the problem of mediation queries evolution. One limitation of the majority of the data integration systems is related to the capability of evolving according to dynamic information systems. The work presented in [Nica

et al. 1999] is one of a few to study the view adaptation problem in dynamic information integration systems proposing the Evolvable View Environment (EVE) framework as a generic approach to solve issues related to view evolution under schema changes for both view definition adaptation and view extent maintenance after synchronization. Such work adopts the relational model as the common model. One limitation of EVE is that it considers only the evolution of the mediation queries in function of the data sources evolution, i.e., it does not discuss how mediation queries must evolve in response of user requirements evolution.

It is well known that in the GAV approach it is hard to guarantee the extensibility of the data integration system [Halevy 2000, Levy 2000, Ullman 1997]. Consequently, to have a system that preserves the advantages of the query reformulation proposed by the GAV approach is crucial to provide a solution for the maintenance of the mappings between the mediation schema and the data source schemas. In this work, we propose a solution for mediation queries evolution in response of user requirements and data source schemas update in the context of XML-based data integration systems.

In the next chapter, we will present an overview of the data integration system proposed in our work by describing its main components.

Chapter 3

Architecture Overview

3.1 Introduction

This work proposes a mediator-based data integration system that offers an integrated view of data distributed in several autonomous and heterogeneous data sources [Lóscio et al. 2001, Lóscio et al. 2002a]. One distinguishing feature of the proposed system is that it presents solutions for the problems concerning to mediation queries generation and maintenance in dynamic environments.

The system also combines features of both approaches for data integration supporting the execution of virtual and materialized queries. This is done to minimize the impacts of most common problems presented by the mentioned approaches. Some portions of data more intensively unavailable and static may be materialized in a data warehouse and the more dynamic data are accessed by virtual queries. It also uses a cache system in order to answer the most frequently asked queries. All these resources are put together with the goal of improving the overall query response time.

As shown in Figure 3.1, the system architecture can be divided into four spaces:

- *Common core*: this space feeds the mediator generation and maintenance space with information about data sources schemas while receiving data source queries from the data integration space and answering them.

- *Data integration space*: the main component of this space is the mediator which is responsible for restructuring and merging data from autonomous data sources and for providing an XML integrated view of distributed data. Other components of this space are used to improve the overall query response time of user queries.

- *Mediation queries generation and maintenance space*: this space executes the mediation

queries generation and maintenance, i.e., it generates and maintains the consistency of the mappings between the mediation elements and the source elements, which are used during the user queries execution. The process of mediation queries generation is based on the approach for discovering relational view expressions proposed by Kedad & Bouzeghoub [Kedad et al. 1999]. Since we adopted XML as the common data model, we had to adapt this approach in order to generate XML-based mediation queries. The mediation queries maintenance is executed by a global evolution process, which receives events about data source schemas changes and users' requirements changes, and propagates them into the mediation level.

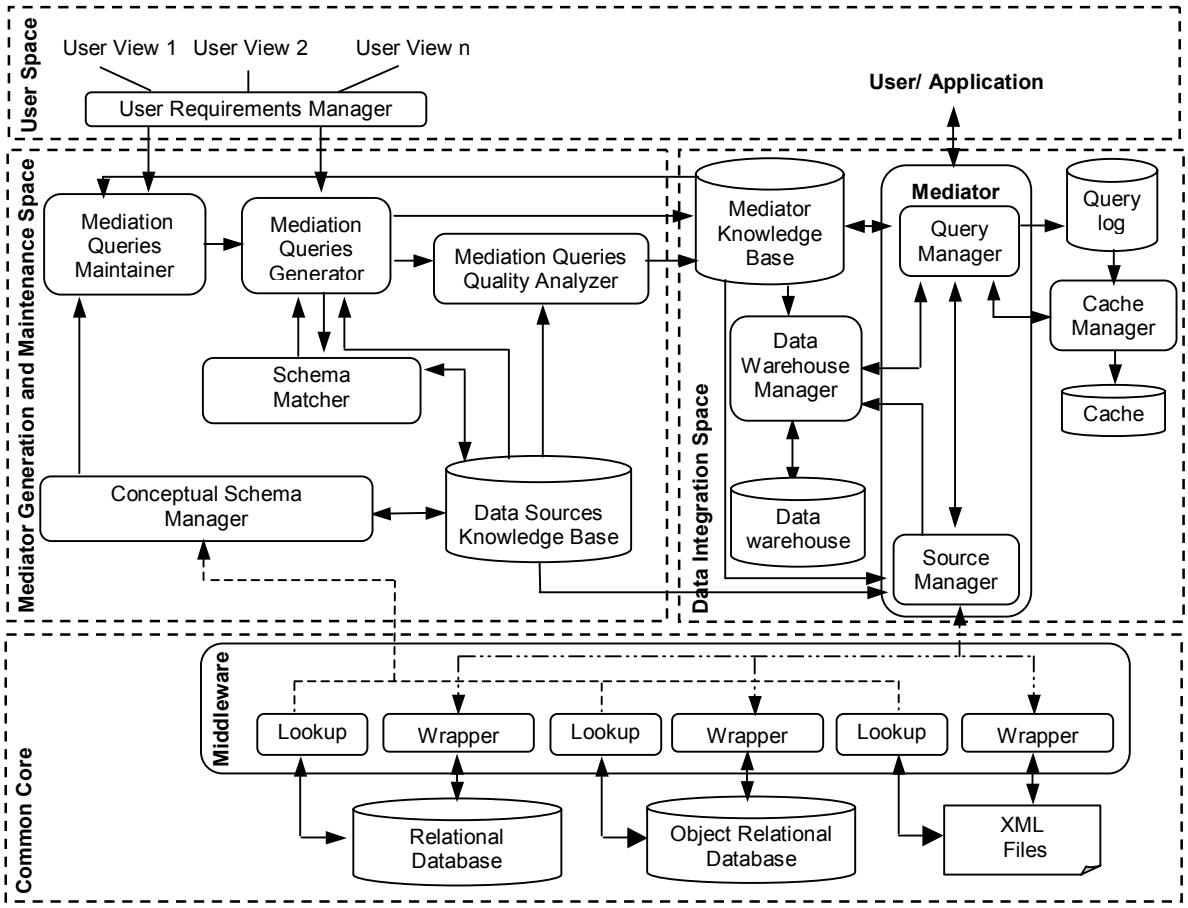

Figure 3.1 – Architecture overview

▪ *User space*: this space is composed by the *Users' Requirements Manager*, which offers an interface for the users' requirements definition. Besides, it is responsible for notifying the *Mediation Queries Maintainer* about the users' requirements changes.

In the following, we present an overview of our system based on these spaces.

3.2 Common core

This space is related to the *Mediator Generation and Maintenance Space* and the *Data Integration Space*, and is composed by the following components: the data sources, the wrappers and the middleware.

- *Data sources*

The data sources are heterogeneous, autonomous and dynamic. This is due to the fact that the data sources support local applications and update their data and schemas independently, possibly without any concern of how this may affect the data integration system based upon them. Data sources may also be added to the system, or become temporally or definitively unavailable. A data source is included in the integration system via a wrapper and a lookup process that serve as bridges between the data source and the other components of the system. When a data source joins the system, it publishes its exported schema describing the information available through this data source. It is important to note that when a data source changes its exported schema it is necessary to publish it again. The exported schema must be updated to reflect corresponding data source schemas changes or when some information needs to be added or dropped from it. The data sources ideally publish the most recent version of their exported schema in order to keep the consistency between the information available to the user and the data actually stored in the data source.

- *Middleware*

The middleware offers two main services (translation of data and queries, and extraction of exported schemas), which are executed by the following modules:

  – *Wrappers*

  Wrappers are necessary for each data source to translate application queries into specific source queries and to translate the data returned by the local data sources into the common data model [Hammer et al. 1998]. Due to the flexibility of XML to represent both structured and semi-structured data there is an increasing interest in using XML as a common data model for data exchange and integration [Baru et al. 1999, Draper et al. 2001, Gardarin et al. 2002, Ives et al. 1999]. In this work, we adopt XML as the common data model.

  Several works have proposed solutions for the translation of conventional data, mainly relational data, to XML [Baru 1999, Fernandez et al. 2000, Shanmugasundaram et al. 2001, Vittori et al. 2001]. The work presented in [Braganholo 2002] proposes a language that

allows the specification of updatable XML views over relational databases. In [Carey et al. 2000] it is discussed the problem of defining XML views in the context of object-relational data.

Considering the increasing use of XML as the standard format for data exchange and representation, in the future, data sources interested in sharing their data will naturally offer XML views of them.

It is important to observe that wrappers are responsible for the translation of data and queries, i.e., wrappers are not responsible for the translation of schemas to a common data model nor for the extraction of metadata on local data sources.

– *Lookup*

This module is responsible for the extraction of the exported schemas from the local data sources. To do this, each *Lookup* sends a request to its corresponding data source and as a result it receives the corresponding exported schema defined in the data format of the remote source.

When a data source joins the system, the *Lookup* requests its exported schema describing the information available through this data source. Whenever a data source changes its exported schema it is necessary to extract it again. However, as mentioned earlier the data sources are autonomous and may update their schemas independently from the data integration system. Hence, the *Lookup* must execute the extraction of the exported schemas periodically in order to keep the consistency between the information available to the user and the data actually available in the data source. To do this, the system maintains a log file associated with each data source which includes, for example, an entry for each data source schema update. The schema update frequency will be used to determine when the lookup must extract the data source schema. When the *Lookup* extracts an exported schema defined in its own data model (ex: relational or object-oriented) it also translates the schema to XML Schema. After this, it sends the translated schema to the *Conceptual Schema Manager*.

3.3 Data integration space

The main component of this space is the mediator which is responsible for the activities of decomposing user queries into subqueries over the underlying data sources and integrating the corresponding results. Originally, our data integration system was proposed to adopt only the virtual approach to data integration. However, in [Batista et al. 2003] we extended the system's architecture with other components with the goal of optimizing the performance of user queries

execution. The extended architecture combines features of both data integration approaches supporting the execution of virtual and materialized queries. This is done to minimize the impacts of most common problems presented by the mentioned approaches. Some portions of data more intensively unavailable and static may be materialized in a data repository and the more dynamic data are accessed by virtual queries.

Another distinguishing feature of the extended architecture is the use of a local cache, i.e., a repository to store prepared answers for the most frequently queries submitted to the data integration system. More details about the specification and implementation of the modules responsible for the optimization of query response time can be found in [Batista 2003]. In the following, we describe the components of the data integration space in more details.

- *Mediator*

A *Mediator* is a software device supporting an integrated view of several data sources. A schema for the integrated view is available from the *Mediator*, thus allowing queries to be made against that schema. The mediation schema contains all the elements needed to answer the users queries which can be computed from the data sources. Also, the elements in the mediation schema are associated with mediation queries, which are responsible for their computation.

The *Mediator* is composed by two sub-modules:

– *Query Manager*

Whenever a query is submitted to the data integration system, the *Query Manager* first analyzes the query to determine where the data which is relevant to its answer is stored. If the query results are stored in cache, they will promptly be returned by the *Query Manager*. On the contrary, if these results are not stored in cache, then it is necessary to identify where the elements which compose the query results are stored. These elements can be virtual or materialized. Virtual elements are accessed from the data sources and materialized elements are obtained directly from the data warehouse. After retrieving the data, the *Query Manager* composes the results and returns the answer to the user.

– *Source Manager*

One of the tasks of the Source Manager is to interact with the data sources, sending queries addressed to the source wrappers and returning the corresponding results to the Query Manager. The Source Manager also monitors the sources in order to determine if there are portions of data that may be materialized in the data warehouse [Harinarayan et al. 1996, Gupta et al. 1997, Theodoratos et al. 1997]. This procedure implies in analyzing the data sources through their metadata and defining reference values for some materialization

criteria, for example: availability and update frequency of the data source. More details about the criteria adopted to select the portions of data to be materialized may be found in [Batista 2003].

- *Cache Manager*

The *Cache Manager* is responsible for the maintenance of a locally stored cache with respect to space availability, substitution policies and contents refreshment [Dar 1996]. Another task of the *Cache Manager* is to periodically access the *Query Log*, to verify the frequencies of submitted queries and to identify if there are new queries results to be stored in cache. In this case, the *Query Manager* will recompute the new queries and store their results in the cache. Periodically all the cached queries must be recomputed to proceed with the refreshment of cache contents.

- *Data Warehouse Manager*

The main role of this module is the maintenance of the data warehouse with respect to the materialization of the data suggested by the *Source Manager* and refreshment of the materialized data. Data warehouse maintenance policies were investigated in previous works [Gupta et al. 1995, Widom 1995, Rundensteiner et al. 2000]. These policies addressed the problem of keeping the consistence of the data warehouse with sources contents. As we already discussed, this can be done either by overriding the existing contents or appending new data to existing data in the data warehouse.

The *Data Integration Space* has two additional data repositories: the data warehouse, which stores data more intensively unavailable and static, and the cache, which stores prepared answers for the most frequently queries submitted to the integration system. It is important to note that the construction of the cache and the data warehouse will be done in two different phases:

– The data warehouse is built during the initial construction of the integrated view and hereafter it will be maintained by the *Data Warehouse Manager*. The materialization of data to be stored in the data warehouse is done through the execution of mediation queries. During the refreshment of the data warehouse, the elements must be recomputed, i.e., the corresponding mediation queries must be re-executed.

– Initially, at system startup, the cache is empty. The cache will be populated at runtime when queries are submitted to the system. This will be done based on the value of the queries frequency, i.e, the queries posed to the system more frequently will have their answers stored in cache.

3.4 Mediation queries generation and maintenance space

The main goals of this space are the generation and maintenance of the mediation queries. As mentioned earlier, the mediation queries express how to obtain the data for the elements that compose the mediation schema. Besides these tasks, the components of this space are also responsible for managing the exported schemas and identifying the correspondences among them, as described in the following.

- *Conceptual Schema Manager*

To provide a high-level abstraction for information described by an *XML Schema* we propose the X-Entity data model. The X-Entity model is not a new formalism for conceptual modeling, rather it is an extension of the Entity Relationship (ER) model [Chen 1976], i.e., we use some basic features of the ER model and we extend it with some additional features to better represent *XML Schemas*. The X-Entity model will be described in Chapter 4 along with the conversion process of *XML Schemas* to X-Entity schemas.

The *Conceptual Schema Manager* is the module responsible for translating *XML Schemas*, received from the *Lookup* module, to X-Entity schemas. Besides, this module has another task concerning the identification of source schema changes. When the system receives a new version of a given exported schema, it compares the new version of the corresponding conceptual model with the older one, currently stored in the *Data Sources Knowledge Base*, in order to identify schema modifications. The result of this comparison is a set of events specifying data source schema changes. These events are sent to the *Mediation Queries Maintainer*, which propagates them into the mediation queries.

- *Schema Matcher*

This module is responsible for the matching [Rahm et al. 2001] of schemas in order to identify the correspondences among their elements. The problem of schema matching has been broadly investigated in the literature [Miller et al. 1994, Milo et al. 1998, Rahm et al. 2001]. [Rahm et al. 2001] defines the match operation as a function that takes two schemas S_1 and S_2 as input and returns a mapping between them as output, called match result. Each mapping element of the match result specifies that certain elements of schema S_1 logically correspond to, i.e. match, certain elements of S_2.

Recently, various work discussing the integration of XML schemas have been proposed [Benevantano et al. 2001, Madhavan et al. 2001, Melo et al. 2001]. We do not adopt a specific methodology to identify the correspondences between the X-Entity schemas. However, in

Chapter 5, we present a formalism to define the correspondences between elements of two X-Entity schemas.

- *Mediation Queries Generator*

The process of mediation queries generation is based on the approach proposed by Kedad & Bouzeghoub [Kedad et al. 1999], which defines a solution space providing the set of potential queries corresponding to a given entity in the mediation schema. Such queries specify how to obtain the integrated instances of a given mediation entity from the source data. We adapted this approach in order to generate XML-based mediation queries. The process of mediation queries generation can be summarized in three main steps: i) selection of relevant sources which potentially allow to compute a given mediation element, ii) identification of possible operators to apply between different sources and iii) generation of all possible queries from the selected sources and operators.

- *Mediation Queries Maintainer*

This module is responsible for the propagation of source schema changes and users' requirements evolution. The evolution of the user needs consists in adding, removing or modifying a user requirement. These changes impact the mediation schema by adding, modifying or deleting an element from the mediation schema. If these changes can be reflected in the mediation queries, the modifications on the mediation schema are committed; otherwise the user is informed that his new requirements cannot be satisfied.

The evolution of the data source schemas consists in adding or removing a data source or modifying the schema of a data source participating in the data integration system. If one of theses changes occurs, it has to be propagated to the mediation queries. The mediation queries are modified if the source elements on which they were defined are modified or when a source element is added or deleted.

To propagate a data source schema change or a user requirement change, the *Mediation Queries Maintainer* uses a set of event-condition-action (ECA) rules where: the event is a schema operation which represents the change, the condition is related either to the local schemas or to the mediation schema metadata, and the action is a sequence of propagation primitives which update the metadata describing the mediation queries. More details about the propagation of schema changes into the mediation queries will be presented in Chapter 6.

- *Mediation Queries Quality Analyzer*

Besides the management of the mediation queries evolution, it is also important to evaluate the impact of the changes (at the source level or at the user level) on the quality of the

integrated data. Indeed, it is easy to understand that some changes may lead to a need for modifying the mediation queries in such a way they select more or less data from the sources. Consequently, it becomes important to qualify the impact of changes either with respect to the users' requirements (the ratio of queries which remain computable after a given change) or with respect to their expectation (to what extent a given change impacts the semantics of a given query, i.e. the data delivered by a given query).

The *Mediation Queries Quality Analyzer* evaluates the impact of the schema changes propagation on the general quality of the system. We propose to evaluate the variation of the quality of the data integration system after the propagation of a source schema or a user requirement change. We consider that the quality of a data integration system corresponds to the overall quality of the mediation queries. A set of quality criteria [Wang et al. 1996, Naumann et al. 1999] (reputation, reliability, availability and response time, for example) may be used to calculate the quality score of a given mediation query.

The *Mediator Generation and Maintenance Space* has two additional knowledge bases: i) the *Data Sources Knowledge Base*, which stores data source descriptions, including the exported schemas defined in XML Schema and the correspondence assertions specifying the relationships between their elements. The data sources descriptions are important for the mediation queries generation and maintenance, and for translating application queries into precise query plans, and ii) the *Mediator Knowledge Base*, which stores mediator descriptions, including the mediation schema defined in XML Schema and the mediation queries.

3.5 User space

This space is composed by the *User Requirements Manager* which has an interface for the user's requirements specification. This interface offers information about the domain being modeled in order to facilitate the user's work. The users' requirements are stored in an X-Entity schema, which will be used as the basis for the mediation schema definition process. Whenever a new data source joins the system or new information is added into an existing data source schema, then the users' requirements are analyzed to identify the user entities that become computable. Such entities are inserted into the mediation schema and their corresponding mediation queries are generated.

Since the users' requirements continue to evolve it is also important to propagate the users' requirements changes into the mediation schema and the mediation queries. When the *Users' Requirements Manager* receives a new requirement or a change requirement, it identifies the change operations to be performed in the mediation schema in order to reproduce them. After,

it sends these changes to the *Mediation Queries Maintainer*, which will propagate them to the mediation queries. If the new requirement can be computed from the data available in the data sources then the change operation in the mediation schema will be confirmed; otherwise it will be cancelled.

3.6 Concluding remarks

In this chapter, we presented an architectural overview of the data integration system proposed in our work. The system is composed by four main spaces: i) *Common Core*: performs the interaction with the data sources, ii) *User Space*: is the interface with the users, iii) *Mediation Queries Generation and Maintenance space*: performs all the activities concerning both mediation queries generation and mediation queries maintenace and iv) *Data Integration Space*: is responsible for the activities concerning the execution of queries submitted to the system. The focus of this work is the Mediation Queries Generation and Maintenance space.

One distinguishing feature of the proposed system is that it presents solutions for the problems concerning mediation queries generation and maintenance. Our approach for mediation queries produces mediation queries which can be easily updated in response of source schema and user requirements modifications. In some work [Chawathe et al. 1994, Baru et al. 1999], the mediation queries are generated manually, which can be a very time-consuming and error prone task. In other work [Arens et al. 1993, Bergamaschi et al. 1998], the mediation queries are automatically generated through algorithms. The work proposed in [Ambite et al. 2001] presents a detailed description of the process of generating integration axioms, which are very similar to the mediation queries proposed in our work. None of these work, use XML as the common data model. As the structure of XML data is more flexible than the structure of conventional data, then the process of computing mediation elements is more complex. In the relational model, for example, a relation is composed by a set of tuples having the same schema and consisting of attributes, which are atomics and monovalued. On the other hand, XML elements may be composed by other elements and attributes, and elements with the same type may have different structures. Other systems [Ives et al. 1999, Draper et al. 2001, Gardarin et al. 2002, Cali et al. 2003] do not discuss how the mediation queries can be specified or obtained.

The problem of mediation queries evolution is discussed in some work [Nica et al. 1999, Ambite et al. 2001, McBrien et al. 2002]. The work presented in [Ambite et al. 2001] adopts an approach similar to ours for defining mediation queries. As proposed in our work, they also produce mediation queries which can be easily updated in response of source schema

modifications. In [McBrien et al. 2002] is presented an approach to handle both schema integration and schema evolution in heterogeneous database architectures, but instead of mediation queries they use primitive transformations to automatically translate queries posed to the global schema to queries over the local schemas. The work presented in [Nica et al. 1999] is one of a few to study the view adaptation problem in dynamic information integration systems proposing the Evolvable View Environment (EVE) framework [Rundensteiner et al. 1997] as a generic approach to solve issues related to view evolution under schema changes. They propose several algorithms to find SQL queries rewritings for views that become invalid after a data source schema change. In contrast to our approach, EVE uses materialized views for data integration. None of the propose approaches consider the problem of evolving mediation queries when users' requirements are inserted, modified or removed. We propose an incremental approach to develop the mediation schema and the mediation queries [Lóscio et al. 2002b, Bouzeghoub et al. 2002] based on the evolution of the data source schemas and the evolution of the users' requirements.

Similar to the data integration systems presented in [Baru et al. 1999, Ives et al. 1999, Draper et al. 2001, Gardarin et al. 2002], we use XML as the common model for data exchange and integration. Other key issue of our proposal is the use of XML Schema as the language for representing the mediation schema and the exported schemas.

In the next chapters, we discuss in more details our research contributions. Chapter 4 presents the X-Entity model and the process of converting an *XML Schema* to its corresponding X-Entity schema. Chapter 5 describes our approach for generating XML-based mediation queries and Chapter 6 introduces our solution to manage the evolution of the mediation queries.

Chapter 4

Conceptual Modeling of XML schemas

4.1 Introduction

XML documents [Bray et al. 2000] have been widely used for interchanging data between heterogeneous systems. An XML document is composed by one or more nested elements, which may contain other elements, called subelements, or may contain character data, optionally mixed with subelements. Additionally, an element may have a set of attribute specifications. To better describe the structure and content of XML data, several XML schema languages have been proposed: DTD, XML Schema [Fallside 2001], XDR [Frankston et al. 1998], SOX [Davidson et al. 1999], Schematron [Jellife et al. 2000] and DSD [Klarlund et al. 2000]. Using an XML schema it is possible to specify several constraints on an XML document, for example whether the elements and attributes are required or optional and which element and attribute types are allowed.

This chapter presents X-Entity [Lóscio et al. 2003], a conceptual data model for XML schemas. The X-Entity model is not a new formalism for conceptual modeling, rather it is an extension of the Entity Relationship model [Chen 1976], i.e., it uses some basic features of the ER model and extends it with to better represent XML schemas. The main concept of the X-Entity model is the entity type, which represents the structure of XML elements composed by other elements and attributes. In the X-Entity model, relationship types represent element-subelement relationships and references between elements. The X-Entity model also provides a graphical representation for

XML schemas and it may be used either during the conceptual design of an XML schema or to provide high-level representations for existing XML schemas.

We also present the process of converting an *XML Schema* to an X-Entity schema. This process is based on a set of rules that considers element declarations and type definitions of an *XML Schema* and generates the corresponding conceptual elements.

This chapter is organized as follows. Section 4.2 reviews the ER model and presents the X-Entity model. Section 4.3 introduces basic concepts of the XML Schema language and presents the notation proposed to describe *XML Schemas*. Section 4.4 describes the process of converting an *XML Schema* to an X-Entity schema. Finally, section 4.5 discusses related works and concluding remarks.

4.2 A conceptual model for representing XML schemas

In this section, initially, we discuss the ER model, which is the basis for the X-Entity model, and next, we present the X-Entity model terminology.

4.2.1 Basic concepts of the ER model

The basic concepts of the ER model are entity types and relationship types [Chen 1976]. An entity type describes a collection of entities that have the same properties, called attributes, and it is identified by a name. In ER diagrams, entity types are represented as a rectangular box and attributes are represented by ovals attached to their entity type. Entities of an entity type may be associated with a key constraint. Attributes defined as key attributes have values that are distinct for each individual entity in the collection. In ER diagrams, key attribute names are underlined. Each attribute of an entity type is associated with a domain, which specifies the set of values that may be assigned to that attribute for each individual entity. Attributes are considered as single-valued when they have one value for a particular entity and they are considered as multivalued when they can have a set of values for the same entity. In the latter case, attributes are shown in double ovals. A relationship type R_j among the entity types E_1, E_2,...,E_n defines a set of associations among entities from these types. In ER diagrams, relationship types are displayed as diamond-shaped boxes, which connect by straight lines the participating entity types. Relationship types usually have certain constraints that limit the possible combinations of entities that participate in the corresponding relationship set. A pair of integer number (min, max) may be associated with each participation of an entity type E_i in a relationship R_j, where $0 \leq min \leq 1$ and $max \geq 1$, which means that each individual entity of E_i must participate in at least min and at most max relationship instances of R_j.

4.2.2 Basic concepts of the X-Entity model

In the following, we present the X-Entity model constructs used to create X-Entity schemas. An X-Entity schema S is denoted by $S = (E, R)$, where E is a set of entity types and R is a set of relationship types.

- *Entity type*: an entity type E_i, denoted by E_i $(\{A_1,\ldots,A_n\},\{R_1,\ldots,R_m\},\{D_1,\ldots,D_k\})$, is made up of an entity name E_i, a set of attributes $A_1,\ldots,A_n$, a set of relationships $R_1,\ldots,R_m$, and a set of disjunction constraints $D_1,\ldots,D_k$.

  An entity type represents a set of elements with a complex structure, composed by attributes and other elements (called subelements). An instance of an entity type is a particular element in the source XML document.

  Each entity type has attributes $\{A_1,\ldots,A_n\}$ that describe it. An attribute A_k represents either an attribute or a subelement, which is not composed by other elements or attributes. Each attribute A_k is associated with a domain, denoted $Dom(A_k)$, which specifies its value set. A_k is also associated with a cardinality, denoted $Card(A_k)=(min,max)$, which specifies the minimum and the maximum number of instances of A_k that can be related with an instance of E_i. As in the ER model, the X-Entity model also makes distinction between key attributes and non-key attributes.

  In X-Entity diagrams, different representations are used to connect attributes to its corresponding entity: dotted lines are used to connect optional attributes ($Card(A_k)=(0,*)$) and thick lines are used to connect required attributes ($Card(A_k)=(1,*)$), where * denotes any number ≥ 1. When multiple occurrences of the same attribute A_k are allowed in a given instance of E_i, then the attribute is represented as a multivalued attribute. It is important to observe that, in the ER model multivalued attributes are used to describe properties that may be associated to a set of values. In the X-Entity model, a multivalued attribute is used to describe that a subelement may occurs multiple times in a given element.

  The professor entity type, presented in Figure 4.1, has three attributes: name, phone and office. name and phone are required attributes while office is optional. Besides, the phone attribute is multivalued, which means that an element professor may have multiple occurrences of the subelement phone. The name attribute is a key of the

professor entity type.

Figure 4.1 - Example of an entity type

- *Relationship type*: a relationship type specifies an association between entity types. In the X-Entity model there are two kinds of relationship:
  - *Containment relationship:* a containment relationship between two entity types E_1 and E_2, specifies that each instance of E_1 contains instances of E_2. It is denoted by $R_j(E_1,E_2,(min,max))$, where R_j is the relationship name and (min,max) defines the minimum and the maximum number of instances of E_2 that can be associated with an instance of E_1. In X-Entity diagrams, containment relationships are displayed as diamond-shaped boxes labeled with contains. The straight lines connecting the relationship with the participating entities are directed from the entity E_1 to the entity E_2.

    Each entity type E_i may be associated with one or more containment relationships $R_j(E_i,E_k,(min,max))$, which describes the element-subelement relationship between E_i and other entity types $\{E_1, \ldots, E_n\}$.

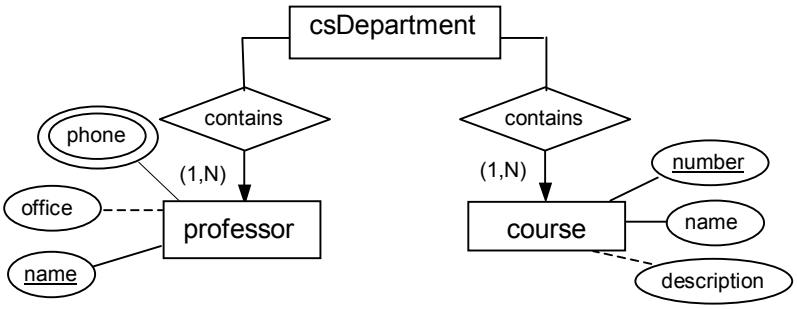

Figure 4.2 - Example of a containment relationship

In the X-Entity schema, presented in Figure 4.2, there are three entity types: csDepartment, professor and course, and two containment relationships. Such relationships specify that an instance of csDepartment has at least one professor subelement and one course subelement, and may have unlimited occurrences of both subelements.

  - *Reference relationship:* a reference relationship, denoted by $R_j(E_1,E_2)$, specifies that the

entity E_1 references the entity E_2. The cardinality of a reference relationship is irrelevant. In X-Entity diagrams reference relationships are represented as a diamond-shaped box labeled with `refers`. The straight lines connecting the relationship with the participating entities are directed to the referenced entity.

The example presented in Figure 4.3 has a reference relationship between `professor` and `course`, which means that an instance of `professor` may be associated with an instance of `course`.

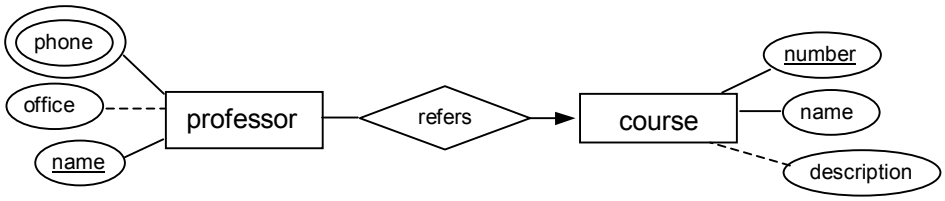

Figure 4.3 - Example of a reference relationship

- *Disjunction constraint*: a disjunction constraint D_k `is` denoted by $D_k((d_1,\ldots,d_n))$, where d_i is an attribute, a containment relationship, a set of attributes or a set of containment relationships of a given entity type E_i. A disjunction D_k defines that an instance of E_i can be associated to only one of the concepts specified in the disjunction constraint. In X-Entity, diagrams for the disjunction constraints are displayed as arcs, which traverses the attributes or relationships participating in the constraint definition. When d_i denotes a set of attributes or a set of containment relationships then the attributes/relationships are attached to a single line, which connects them to the entity.

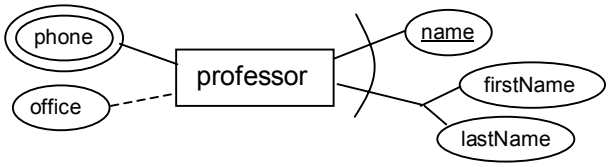

Figure 4.4 - Example of an entity type with a disjunction constraint

The entity type, presented in Figure 4.4, has a disjunction constraint between the attributes `name` and the group composed by the attributes `firstName` and `lastName`, which means that a `professor` instance have either the attribute `name` or the attributes `firstName` and `lastName`.

The `csDepartment` entity type, presented in Figure 4.5, has a disjunction constraint between the `course_name` attribute and the containment relationship between `csDepartment` and `course`, which means that a `csDepartment` instance have either a `course_name` attribute or a `course` subelement.

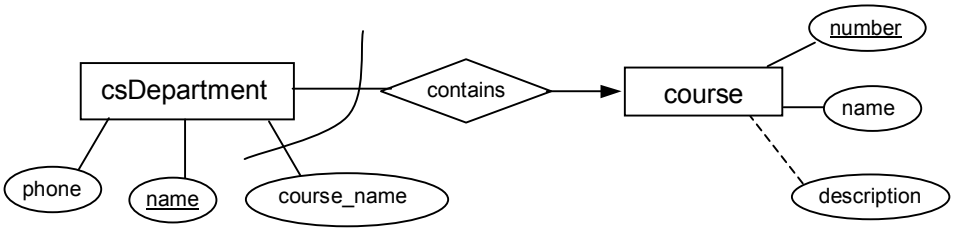

Figure 4.5 – Example of an entity type with a disjunction constraint between an attribute and a relationship

Since XML may be used to represent both structured and semi-structured data, XML schemas are more flexible than conventional database schemas. Therefore, instances of the same entity type may have different structures. To represent this, in an X-Entity schema, both attributes and containment relationships are associated with participation constraints, which represent the occurrence constraints of elements and attributes in an XML document. Besides, disjunction constraints specify restrictions on the participation of two or more subelements in a given element.

4.2.3 XML Schema notation

In this section, we present the notation proposed to describe *XML Schemas*. Initially, we review some basic concepts of the XML Schema language. Consider as an example the `Computer Science Department` schema presented in Figure 4.6.

- **Datatypes**

XML Schema offers two kinds of datatypes: complex types which allow elements in their content and may carry attributes, and simple types which cannot have elements and cannot carry attributes. To define new complex types we use the `complexType` element and to define new simple types we use the `simpleType` element. The `complexType` element must be used when we want to define child elements and/or attributes of an element and the `simpleType` element must be used when we want to create a new type that is a refinement of a built-in type (`string`, `date`, etc). For example, `csDepartmentTy` is defined as a complex type because it is composed by other elements (`courses` and `professors`).

```
<?xml version="1.0" encoding="utf-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
 <xsd:element name="csDepartment" type="csDepartmentTy"/>
 <xsd:complexType name="csDepartmentTy">
  <xsd:sequence>
```

```
   <xsd:element name="courses" type="coursesTy"/>
   <xsd:element name="professors" type="professorsTy"/>
  </xsd:sequence>
 </xsd:complexType>
 <xsd:complexType name="coursesTy">
  <xsd:sequence>
   <xsd:element name="course" type= "courseTy" maxOccurs="unbounded">
     <xsd:key name= "courseNumKey">
       <xsd:selector xpath = "courses/course"/>
       <xsd:field xpath = "number"/>
     </xsd:key>
   </xsd:element>
  </xsd:sequence>
 </xsd:complexType>
 <xsd:complexType name="courseTy">
  <xsd:sequence>
   <xsd:element name="name" type=" xsd:string"/>
   <xsd:element name="number" type=" xsd:string"/>
   <xsd:element name="description" type=" xsd:string" minOccurs="0"/>
  </xsd:sequence>
 </xsd:complexType>
<xsd:complexType name="professorsTy">
  <xsd:sequence>
   <xsd:element name="professor" minOccurs="0" maxOccurs="unbounded"/>
    <xsd:complexType>
     <xsd:sequence>
      <xsd:element name="identification" type="identificationTy"/>
      <xsd:element name="courseNumber" type="string" maxOccurs="unbounded">
        <xsd:keyref name="profCourse" refer="courseNumKey">
          <xsd:selector xpath="professors/professor"/>
          <xsd:field xpath="courseNumber"/>
        </xsd:keyref>
      </xsd:element>
     </xsd:sequence>
     <xsd:attribute name="level" type=" xsd:string" use="required"/>
    </xsd:complexType>
   </xsd:sequence>
 </xsd:complexType>
<xsd:complexType name="identificationTy">
  <xsd:sequence>
   <xsd:choice>
    <xsd:group ref="identity">
    <xsd:element name="name" type=" xsd:string"/>
   </xsd:choice>
   <xsd:element name="phone" type=" xsd:string"/>
   <xsd:element name="office" type=" xsd:string" minOccurs="0"/>
  </xsd:sequence>
 </xsd:complexType>
  <xsd:group name= "identity"/>
     <xsd:sequence>
        <xsd:element name="firstName" type=" xsd:string"/>
        <xsd:element name="lastName" type=" xsd:string"/>
     </xsd:sequence> </xsd:group>
</xsd:schema>
```

Figure 4.6 - Computer Science Department schema

- **Definitions vs. Declarations**

In XML Schema, there is also a distinction between definitions, which create new types, and declarations, which enable elements and attributes with specific names and types (both simple and complex) to appear in XML document instances. Typically, a complex type definition contains a set of element declarations and attribute declarations. Elements and attributes are declared using the element `element` and the `attribute` element respectively. For example, `courseTy` is defined as a complex type, and within the definition of `courseTy` there are three element declarations (`name`, `number` and `description`). Consequently, any element appearing in an instance document whose type is declared to be `courseTy` must consist of three elements, which must be called `name`, `number` and `description` as specified by the values of the declarations' `name` attribute, and the elements must contain a `string` as specified by the values of the declarations' `type` attribute.

- **Basic forms of declaring elements/ attributes**

In XML Schema there are basic forms of declaring elements/ attributes: i) an element/attribute has a type definition inlined in its declaration or ii) an element/ attribute references a type in its declaration. In other words, an element declaration can have a `type` attribute, or a `complexType`/`simpleType` child element, but it cannot have both a `type` attribute and a `complexType`/`simpleType` child element. In the same way, an attribute declaration can have a `type` attribute or a `simpleType` child element, but it cannot have both a `type` attribute and a `simpleType` child element. To illustrate this, consider the declaration of the element named `professor`, which has a complex type definition inlined in its declaration. In contrast, all other element declarations use the attribute `type` to identify the type, regardless of whether the type is simple or complex. It is important to note that all attribute declarations reference simple types, because, unlike element declarations, attributes cannot contain other elements or other attributes.

- **Global Elements and Attributes**

Global elements and global attributes are created by declarations that appear as direct children of the `schema` element. Once declared, a global element or a global attribute can be referenced in one or more declarations using the `ref` attribute. A declaration that references a global element enables the referenced element to appear in the instance document in the context of the referencing declaration. The declaration of a global element also enables the element to appear at the top-level of an instance document. In our example, there is just one global element (`csDepartment`), which will be the root element of the instance documents of the `Computer Science Department` schema.

- **Indicators of occurrences**

In XML Schema, it is also possible to define the minimum and the maximum number of times an element may appear in a document. This is done using the `minOccurs` attribute and the `maxOccurs` attribute respectively. The value of the `maxOccurs` attribute may be a positive `integer` or the term `unbounded` to indicate that there is no maximum number of occurrences. The default value for both the `minOccurs` and the `maxOccurs` attributes is "1". If both attributes are omitted, the element must appear exactly once. For example, the `description` element is optional within `courseTy` because the value of the `minOccurs` attribute in its declaration is "0". Besides this, it may not occur more than once because the attribute `maxOccurs` was omitted. Unlike elements, attributes may appear once or not at all, but no other number of times. Attributes can be declared with an `use` attribute to indicate whether the attribute is `required`, `optional`, or even `prohibited`. For example, the `level` attribute is required.

- **Content Models**

In XML Schema, the content model of a complex type definition is composed by groups of elements. To define groups of elements, XML Schema supports the definition of compositors. In fact, compositors define group of particles, which can also be groups of elements or other compositors. For example, the `sequence` compositor defines ordered groups of elements. The `choice` compositor describes a choice between several possible elements or groups of elements. Another option for constraining elements in a group is to define a `all` compositor which allows all the elements in the group to appear once or not at all, and they may appear in any order. XML Schema also enables groups of elements and attributes to be defined and named, so that they can be used to build up the content models of complex types. To define a group of elements it is used the element `group` and to define a group of attributes it is used the element `attributeGroup`. It is important to note that it is not possible to inline the group definitions. Instead, you must define a reference to a group using the `ref` attribute.

In our example, the `IdentificationTy` has a sequence group, which has four children: a choice group and three element declarations. The choice group element has two children: one child is an inner group element that references the named group `identity` (consisting of the element sequence `firstName` and `lastName`) and the second child is a `name` element declaration. Therefore, in an instance document, the `identification` element must contain either a `firstName` element followed by a `lastName` element or a single `name` element. The choice group is followed by the `phone`, `office` and `email` element declarations. The

consequence of these various groups is that: i) the name element must be followed by a phone, office and email elements in that order or ii) the elements firstName and lastName (in that order) must be followed by a phone, office and email elements in that order. It is important to note that named and unnamed groups that appear in content models (represented by group and choice, sequence, all respectively) may carry minOccurs and maxOccurs attributes.

- **Key and keyref definitions**

XML Schema enables us to indicate that any attribute or element value must be unique within a certain scope and cannot be set to nil. This is done using a key definition, which selects a set of elements to be the scope of the key and specifies the attributes or elements that have to be unique and cannot be set to nil. To define a key constraint it is used the key element, which may have the following subelements: selector and field. The first specifies the scope where the key definition must be valid and the second is used to specify the attributes or elements that compose the key. The name, which is associated to a key definition, makes the key referenceable from elsewhere. To reference a key it is used a keyref definition. In our example, there is only one key definition named courseNumKey, which means that the value of the element number must be unique, i.e., every course element must have a different value for the number element. To ensure that the professor elements are associated with valid course numbers, then the courseNumber of those elements must reference the courseNumKey key. The declaration of courseNumber as a keyref does not mean that its value must be unique, but it does mean that there must exist an element course with the same value for the element number.

In the following, we present the notation to describe *XML Schemas*. Basically, an *XML Schema* is a set of element declarations, complex type and named group definitions. We use the notations: "|" for choice, "?" for zero or one occurrence, and "*" for one or more occurrences.

- *XML Schema*: an *XML Schema* S is denoted by S = ((Complex Type Definition | Element Declaration| Group Definition | Attribute Group)*).
- *Complex type definition*: a complex type definition C_T is a pair with C_T = (name, content), where name defines the name of the complex type and content = ((Element Group Reference | Compositor)?,(Attribute Declaration | Attribute Group Reference)*).
- *Element Group definition*: an element group definition G_E is a three tuple with G_E = (name,(minOccurs, maxOccurs), content), where name defines the name of the

group, (minOccurs,maxOccurs) specify how many times the group can occur in document instances and content is a compositor.

- *Compositor*: a compositor T is a three tuple with T = (constraint,(minOccurs, maxOccurs),content), where constraint[1] indicates the constraint applied to the particles of the compositor, (minOccurs, maxOccurs) [2] specify how many times the compositor can occur in document instances and content[3] =((Element Declaration | Element Group Reference | Compositor)*).

- *Attribute group definition*: an attribute group definition G_A is a pair with G_A = (name, content), where name defines the name of the group, content = ((Attribute Declaration | Attribute Group Reference)*).

- *Element Group reference*: an element group reference G_{Er} is specified by the name of the element group referenced, i.e., G_{Er} = (name), where name defines the name of the element group referenced.

- *Attribute Group reference*: an attribute group reference G_{Ar} is specified by the name of the attribute group referenced, i.e., G_{Ar} = (name), where name defines the name of the attribute group referenced.

- *Element declaration*: an element declaration L_D is a three tuple with L_D = (name, type,(minOccurs, maxOccurs)), where name defines the name of the element, type indicates the type of the element (it can be the name of a complex type or a complex type definition) and (minOccurs,maxOccurs) specify how many times the element can occur in document instances.

- *Attribute declaration*: an attribute declaration A_D is a three tuple with A_D = (name, type, use), where name defines the name of the attribute, type indicates the type of the attribute (it can be the name of a simple type, a simple type definition or the name of a base type) and the attribute use specifies if an attribute is optional or required.

- *Simple type definition*: a simple type definition S_T is a pair with S_T = (name, ((type, facet*))), where name defines the name of the simple type, type specifies the name of

[1] The possible values of the constraint property are: sequence, choice and all. A compositor with a choice constraint defines a group of mutually exclusive elements. A compositor with a sequence constraint defines an ordered group of elements. A compositor with an all constraint describes an unordered group of elements.
[2] The number of occurrences cannot be defined when a compositor is used within a group.
[3] The content of a compositor with an all constraint can be composed only by element declarations.

the base type from which the simple type is derived and `facet` is an element that constraints the range of possible values.

- *Key definition*: a key definition K_D is a three tuple with K_D = (name, selector, field*), where `name` defines the name of the key, `selector` is the name of the element that defines the scope of the key and `field` is the name of an element or attribute which is restricted by the key definition.
- *Keyref definition*: a keyref definition F_D is a four tuple with F_D = (name, key, selector, field*), where the attribute `name` defines the name of the keyref, `key` is the name of referenced key, `selector` is the name of the element that defines the scope of the keyref and `field` is the name of an element or attribute which is restricted by the keyref definition.

We use the "." notation to denote values of properties of the *XML Schema* components. For example, we use the expression L_D.name to obtain the value of the attribute `name` of an element declaration L_D.

4.3 Generating conceptual schemas from XML Schemas

In this section, we describe how to convert an *XML Schema* to an X-Entity schema. The resulting X-Entity schema must hide irrelevant structures represented in the *XML Schema* while describe semantically relevant structures and the relationships among them. Considering the data integration context, understanding the semantic meaning of the represented data is a crucial task.

The conversion process is based on a set of rules for generating conceptual elements (entity types and relationship types) from element/attribute declarations and complex type definitions. The two main tasks of the conversion process are: i) Pre-processing and ii) Conversion. In the first, the *XML Schema* is redefined in order to eliminate some technical and useless characteristics that make it difficult the conversion process. In the second, the pre-processed schema is converted to its corresponding conceptual schema. In this section, consider as example the `Computer Science Department` schema presented in Figure 4.6.

4.3.1 Task 1: Pre-processing

During this task, an *XML Schema* is modified in order to eliminate group references, anonymous type definitions and irrelevant element declarations. These tasks are executed to facilitate the next task of conversion of an *XML Schema* to its corresponding X-Entity schema.

- **Substitution of references**

XML Schema enables groups of elements/attributes to be defined and named, so that the elements/attributes can be used to build up the content models of complex types. The substitution of both named element groups and attribute groups references for their corresponding definition is needed in order to obtain whole complex type definitions. Besides group references, global element references must also be replaced by their corresponding element declarations. Consider, for example, the `identificationTy` complex type presented in Figure 4.7(a), which references a group named `identity`. To obtain a whole description of `identificationTy` the group reference must be replaced by the content of the `identity` group (Figure 4.7(b)).

```
…
<xsd:complexType name="identificationTy">
  <xsd:sequence>
   <xsd:choice>
    <xsd:group ref= "identity">
    <xsd:element name="name" type=" xsd:string"/>
   </xsd:choice>
   …
  </xsd:sequence>
 </xsd:complexType>
 <xsd:group name= "identity">
  <xsd:sequence>
    <xsd:element name="firstName" type=" xsd:string"/>
    <xsd:element name="lastName" type=" xsd:string"/>
  </xsd:sequence>
 </xsd:group> …
```

Figure 4.7 (a) - Identity group definition

```
…
<xsd:complexType name="identificationTy">
  <xsd:sequence>
   <xsd:choice>
    <xsd:sequence>
     <xsd:element name="firstName" type="xsd:string"/>
     <xsd:element name="lastName" type="xsd:string"/>
    </xsd:sequence>
    <xsd:element name="name" type="xsd:string"/>
   </xsd:choice>
   …
  </xsd:sequence>
</xsd:complexType>
…
```

Figure 4.7 (b) – Schema obtained after the substitution of the identity group reference

- **Elimination of anonymous type definitions**

Schemas can be constructed by defining either sets of named types and then declaring elements that reference these types or inlining type definitions in element declarations. In the latter case, anonymous type are originated. In order to facilitate the next conversion task, all anonymous type must be eliminated. Consider, for example, the `professorsTy` complex type presented in Figure 4.8(a), which has a `professor` element declaration. The type of a `professor` element is defined by a complex type whose definition is inlined in its declaration. To eliminate this anonymous type definition a `type` attribute with value `professorTy` must be inserted in the `professor` element declaration and a new complex type named `professorTy` must be created (Figure 4.8(b)).

```
…<xsd:complexType name="professorsTy">
  <xsd:sequence>
   <xsd:element name="professor" minOccurs="0" maxOccurs="unbounded">
    <xsd:complexType>
     <xsd:sequence>
      <xsd:element name="identification" type="identificationTy"/>
      <xsd:element name="courseNumber" type="xsd:string"
                minOccurs="0" maxOccurs="unbounded"/>
     </xsd:sequence>
     <xsd:attribute name="level" type="xsd:string" use="required" />
    </xsd:complexType>
   </xsd:element>
  </xsd:sequence>
 </xsd:complexType>…
```

Figure 4.8(a) – Example of an anonymous type definition

```
…<xsd:complexType name="professorsTy">
  <xsd:sequence>
   <xsd:element name="professor" type="professorTy"
             minOccurs="0"  maxOccurs="unbounded"/>
  </xsd:sequence>
</xsd:complexType>
<xsd:complexType name="professorTy">
 <xsd:sequence>
  <xsd:element name="identification" type="identificationTy"/>
  <xsd:element name="courseNumber" type="xsd:string"
                 minOccurs="0" maxOccurs="unbounded"/>
 </xsd:sequence>
 <xsd:attribute name="level" type="xsd:string" use="required" />
</complexType>…
```

Figure 4.8 (b) – Schema obtained after the creation of the complex type professorTy

- **Elimination of irrelevant element declarations**

Elements just used to compose the structure content of another element must be removed. These elements are semantically irrelevant and they do not need a representation in the conceptual schema. Generally, elements whose type has just one element declaration are semantically irrelevant.

Consider, for example, the `courses` element presented in Figure 4.9(a), which only encapsulates a list of `course` elements. Therefore, the `courses` element declaration must be removed and replaced by the content model of the complex type `coursesTy`. Figure 4.9(b) presents the same complex type definition after the replacement of the `courses` element declaration by the definition of the `coursesTy` complex type.

```
…
<xsd:complexType name="csDepartmentTy">
  <xsd:sequence>
   <xsd:element name="courses" type="coursesTy"/>
   …
  </xsd:sequence>
</xsd:complexType>
<xsd:complexType name="coursesTy">
 <xsd:sequence>
  <xsd:element name="course" type="courseTy"
               minOccurs="0" maxOccurs="unbounded"/>
 </xsd:sequence>
  …
</xsd:complexType>
<xsd:complexType name="courseTy">
  <xsd:sequence>
   …
  </xsd:sequence>
</xsd:complexType>
…
```

Figure 4.9 (a) – Example of irrelevant element declaration

```
…
<xsd:complexType name="csDepartmentTy">
  <xsd:sequence>
   <xsd:sequence>
    <xsd:element name="course" type="courseTy"
             minOccurs="0" maxOccurs="unbounded"/>
   …
   <xsd:sequence>
</xsd:complexType>
<xsd:complexType name="courseTy">
 <xsd:sequence>
  <xsd:element name="name" type="string"/>
  <xsd:element name="number" type="string"/>
  <xsd:element name="description" type="string"
           minOccurs="0"/>
 </xsd:sequence>
</xsd:complexType>
…
```

Figure 4.9 (b) – Schema obtained after the elimination of the complex type coursesTy

In our example, other irrelevant element declaration that must be eliminated is the `identification` element declaration. Since this element stores information about the element `professor` only, then it is not semantically relevant. Thus, its declaration must be removed and its content (specified in the definition of the complex type `identificationTy`) must be inserted in the content of the `professor` element. Figure 4.10(a) presents the definition of the complex type `professorTy` before the removal of the declaration of the element `identification`. Figure 4.10(b) presents the same complex type definition after the

replacement of the element declaration `identification` by the definition of the complex type `identificationTy`.

```
…
<xsd:complexType name="professorTy">
 <xsd:sequence>
  <xsd:element name="identification" type="identificationTy"/>
  <xsd:element name="courseNumber" type="xsd:string"
           minOccurs="0" maxOccurs="unbounded"/>
 </xsd:sequence>
 …
</xsd:complexType>
<xsd:complexType name="identificationTy">
 <xsd:sequence>
  <xsd:choice>
   <xsd:sequence>
    <xsd:element name="firstName" type="xsd:string"/>
    <xsd:element name="lastName" type="xsd:string"/>
   </xsd:sequence>
   <xsd:element name="name" type="xsd:string"/>
  </xsd:choice>
  <xsd:element name="phone" type="xsd:string"/>
  <xsd:element name="office" type="officeTy"/>
 </sequence>
</complexType>
…
```

Figure 4.10(a) - Second example of irrelevant element declaration

```
…
<xsd:complexType name="professorTy">
 <xsd:sequence>
  <xsd:sequence>
   <xsd:choice>
    <xsd:sequence>
      <xsd:element name="firstName"
                   type="xsd:string"/>
      <xsd:element name="lastName"
                   type="xsd:string"/>
    </xsd:sequence>
    <xsd:element name="name" type="xsd:string"/>
   </xsd:choice>
   <xsd:element name="phone" type="xsd:string"/>
   <xsd:element name="office" type="officeTy"/>
  </xsd:sequence>
  <xsd:element name="courseNumber" type="xsd:string"
               minOccurs="0" maxOccurs="unbounded"/>
 </xsd:sequence>
 …
</xsd:complexType>
…
```

Figure 4.10(b) - Schema obtained after the elimination of the element identification

- **Renaming of elements**

Another important task to be executed during the pre-processing module is the renaming of conflicting element declarations. When two elements have the same name but different complex types, then one of them must be renamed. This is done to avoid conflicts during the creation of entity type definitions. Consider the *XML Schema* presented in Figure 4.11 (a), which has two

elements `section` with different complex types (`sectionBookTy` and `sectionMagazineTy`). Thus, one of the `section` elements must be chosen to be renamed. Figure 4.11 (b) shows the exported schema obtained after the renaming of the `section` element in the content model of the complex type `bookTy`.

```
<xsd:schema>
 <xsd:element name="library" type="libraryTy">
 <xsd:complexType name="libraryTy">
  <xsd:sequence>
    <xsd:element name="book" type="bookTy"/>
    <xsd :element name="magazine" type="magazineTy"/>
   </xsd:sequence>
 </xsd:complexType>
 <xsd:complexType name="bookTy">
  <xsd:sequence>
   <xsd:element name="title" type="xsd:string"/>
   <xsd:element name="section" type="bookSectionTy" maxOccurs="unbounded"/>
  </xsd:sequence>
<xsd:complexType name="magazineTy">
 <xsd:sequence>
  <xsd:element name="title" type="xsd:string"/>
  <xsd:element name="section" type="magazineSectionTy" maxOccurs="unbounded"/>
 </xsd:sequence>
</xsd:complexType>…
```

Figure 4.11 (a) – Example of two elements with the same name and different types

```
<xsd:schema>
<xsd:element name="library" type="libraryTy">
<xsd:complexType name="libraryTy">
 <xsd:sequence>
  <xsd:element name="book" type="bookTy"/>
  <xsd:element name="magazine"
               type="magazineTy"/>
 </xsd:sequence>
</xsd:complexType>
<xsd:complexType name="bookTy">
 <xsd:sequence>
  <xsd:element name="title" type="xsd:string"/>
  <xsd:element name="chapter" type="bookSectionTy" maxOccurs="unbounded"/>
 </xsd:sequence>
<xsd:complexType name="magazineTy">
 <xsd:sequence>
  <xsd:element name="title" type="xsd:string"/>
  <xsd:element name="section" type="magazineSectionTy" maxOccurs="unbounded"/>
 </xsd:sequence>
<xsd:/complexType>…
```

Figure 4.11 (b) - Schema obtained after the renaming of one of the elements section

All the information about the pre-processing of *XML Schemas* (ex: removal and creation of element declarations and complex type definitions) must be documented. Figure 4.12 presents the `Computer Science Research` schema after the pre-processing step.

```
<?xml version="1.0" encoding="utf-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
 <xsd:element name="csDepartment" type="csDepartmentTy"/>
```

```
<xsd:complexType name="csDepartmentTy">
 <xsd:sequence>
  <xsd:element name="course" type= "courseTy" minOccurs="0" maxOccurs="unbounded">
    <xsd:key name="courseNumKey">
      <xsd:selector xpath="course"/>
      <xsd:field xpath="number"/>
     </xsd:key>
  </xsd:element>
  <xsd:element name="professor" type= "professorTy" minOccurs="0"
              maxOccurs="unbounded">
    <xsd:keyref name="profCourse" refer="courseNumKey">
      <xsd:selector xpath="professor"/>
      <xsd:field xpath="courseNumber"/>
    </xsd:keyref>
  </xsd:element>
 </xsd:sequence>
</xsd:complexType>
<xsd:complexType name="courseTy">
 <xsd:sequence>
  <xsd:element name="name" type=" xsd:string"/>
  <xsd:element name="number" type=" xsd:string"/>
  <xsd:element name="description" type=" xsd:string" minOccurs="0"/>
 </xsd:sequence>
</xsd:complexType>
<xsd:complexType name="professorTy">
 <xsd:sequence>
  <xsd:sequence>
   <xsd:choice>
    <xsd:sequence>
      <xsd:element name="firstName" type=" xsd:string"/>
      <xsd:element name="lastName" type=" xsd:string"/>
    </xsd:sequence>
    <xsd:element name="name" type=" xsd:string"/>
   </xsd:choice>
  <xsd:element name="phone" type=" xsd:string"/>
  <xsd:element name="office" type=" xsd:officeTystring" minOccurs="0"/>
  <xsd:element name="email" type="string"/>
  <xsd:element name="courseNumber" type=" xsd:string" minOccurs="0"
              maxOccurs="unbounded"/>
 </xsd:sequence>
 <xsd:attribute name="level" type=" xsd:string" use="required" />
</xsd:complexType>
</xsd:schema>
```

Figure 4.12 – Pre-processed Computer Science Department schema

4.3.2 Task 2: Conversion

Assuming that the *XML Schema* has been already pre-processed, it is possible to map it into an X-Entity schema. This is done using a set of rules that were defined to generate X-Entity concepts (introduced in Section 4.2.2) from *XML Schema* concepts. In the following, consider C_T as a complex type definition, L_D and L_D' as element declarations and:

$E_i.A$: the set of attributes of the entity E_i.

$E_i.R_j$: the set of relationships of the entity E_i.

$E_i.D$: the set of disjunction constraints of the entity E_i.

- **Rule 1 - Conversion of elements having a complex type**

An element declaration L_D, where L_D.type is the name of a complex type, originates an entity type E_i, whose name is L_D.name, if there is not an entity type E_i' whose name is L_D.name.

Rule 1 specifies that all element declarations whose type is a complex type, i.e., an element that is composed by other elements and attributes, must be mapped to an entity type. Initially, when the entity type E_i is created, its sets of attributes, relationships and disjunction constraints are empty (E_i (∅,∅,∅)). Later, when the other conversion rules are applied, attributes, relationships and constraints will be added in the definition of the entity type E_i. In the Computer Science Department schema there are three elements with complex types: CsDepartment, Course and Professor. These element declarations originate the following entity type definitions:

- csDepartment(∅,∅,∅), course(∅,∅,∅) and professor(∅,∅,∅)

- **Rule 2 - Conversion of elements having a base type**

An element declaration $L_D \in C_T$.content, where L_D.type is a base type, originates an attribute A_k, whose name is L_D.name, in the set of attributes of the entity E_i ($\underline{E_i.A} = \underline{E_i.A} \cup \{A_k\}$), such that $E_i = L_D'$.name, L_D'.type = C_T.name. The domain of A_k is defined based on the value of the type attribute of L_D (Dom(A_k) = L_D.type) and card(A_k) is determined based on the values of the minOccurs and maxOccurs attributes of the element declaration L_D, as described below:

If L_D.maxOccurs ≥ 1 then card(A_k) = (L_D.minOccurs, L_D.maxOccurs)

If L_D.maxOccurs = "unbounded" then card(A_k)=(L_D.minOccurs,n)

Rule 2 specifies that all element declarations, which have a base type and participate in the content model of a complex type C_T, must be mapped to an attribute in all entity types originated from element declarations whose type is C_T. A complex type definition C_T describes the attributes and relationships associated with all entity types originated from element declarations whose type is C_T.

After the application of the Rule 2 in the Computer Science Department schema, the entity types described below are obtained. For example, the element declaration L_D =(description,string,(0,1)) originates the attribute description. As L_D participates in the content model of courseTy, then the description attribute belongs to the course entity type.

- course({name,number,description},∅,∅)

- professor({firstName,lastName,name,phone, office,courseNumber},∅,∅)

- **Rule 3 - Conversion of attributes**

An attribute declaration $A_D \in C_T$.content originates an attribute A_k, whose name is A_D.name, in the set of attributes of the entity E_i ($\underline{E_i.A} = \underline{E_i.A} \cup \{A_k\}$), such that $E_i = L_D'$.name and L_D'.type = C_T.name. The domain of A_k is defined based on the value of the type attribute of A_D (Dom(A_k) = A_D.type) and card(A_k) is determined based on the value of the attribute use of the attribute declaration A_D as follows: i) if A_D.use = "optional" then card(A_k)=(0,1) and ii) if A_D.use = "required" then card(A_k)=(1,1).

Rule 3 specifies that all attribute declarations must be defined as attributes. Similar to Rule 2, an attribute originated from an attribute declaration $A_D \in C_T$.content must be inserted in the set of attributes of all entities originated from elements with type C_T. In the Computer Science Department schema there is only one attribute declaration A_D=(level, string, required), which participates in the content model of the complex type professorTy ($A_D \in$ professorTy.content). A_D originates the level attribute, which belongs to the set of attributes of the professor entity (the only entity originated from an element whose type is professorTy).

- professor({firstName,lastName,name,phone, office,courseNumber,level},∅,∅)

- **Rule 4 - Creation of containment relationships**

An element declaration $L_D \in C_T$.content, where L_D.type is a complex type, originates a containment relationship R_j(E_i',E_i,(min,max)), such that E_i is an entity type whose name is L_D.name and E_i' is an entity type whose name is L_D'.name and L_D'.type = C_T.name. The value of the cardinality of the relationship is defined based on the values of the minOccurs and maxOccurs attributes of the element declaration L_D, as described below:

If L_D.maxOccurs ≥ 1 then R_j(E_i',E_i,(L_D.minOccurs,L_D.maxOccurs))

If L_D.maxOccurs = "unbounded" then R_j(E_i',E_i,(L_D.minOccurs,n)) where n denotes an unlimited number of occurrences of E_i.

A containment relationship is defined between an entity type E_i originated from an element declaration $L_D \in C_T$.content and all entity types {E_1, …,E_n} originated from element declarations whose type is C_T.

In the Computer Science Department schema there are two containment relationships:

- csDepartment_course(csDepartment,course,(0,N))
- csDepartment_professor (csDepartment,professor,(0,N))

The csDepartment_course relationship, for example, specifies that an instance of csDepartment contains one or more instances of course. The cardinality of the relationship is determined by the values of the attributes minOccurs and maxOccurs of the course entity type.

When a containment relationship R_i is defined between the entity types E_1 and E_2, then the set of relationships associated with the entity E_1 must be updated in order to include the relationship R_j ($\underline{E_1.R} = \underline{E_1.R} \cup \{R_j\}$). In our example, the following definition for the csDepartment entity is obtained:

- csDepartment=(∅,{csDepartment_course,csDepartment_professor},∅)

Consider, for example, the *XML Schema* presented in Figure 4.13(a). The complex type bookTy has an element author with type authorTy. This results in a containment relationship book_author(book,author,(0,N)). In the same way, the complex type authorTy has an element book with type bookTy. This results in another containment relationship author_book(author,book,(0,N)). In this case, as presented in Figure 4.13 (b), only one containment relationship is represented in the X-Entity diagram. In a similar way, it is possible to represent recursive relationships.

```
<xsd:schema>
 <xsd:element name="library" type="libraryTy">
 <xsd:complexType name="libraryTy">
   <xsd:element name="book" type="bookTy"
               minOccurs="1" maxOccurs="unbounded">
 </xsd:complexType>
 <xsd:complexType name="bookTy">
   <xsd:element name="title" type="xsd:string">
   <xsd:element name="author" type="authorTy"
               minOccurs="0" maxOccurs="unbounded">
 <xsd:complexType name="authorTy">
   <xsd:element name="name" type="xsd:string">
   <xsd:element name="book" type="bookTy"
               minOccurs="0" maxOccurs="unbounded">
 </xsd:complexType>…
```

Figure 4.13(a) - Library XML Schema

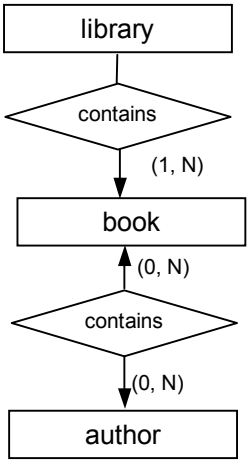

Figure 4.13(b) - Conceptual schema for the Library XML Schema

- **Rule 5 - Conversion of choice groups**

A compositor definition $T \in C_T$.content, where T.constraint = "choice" and T.content = $(t_1,\ldots,t_n)$, originates a disjunction constraint $D_k(d_1,\ldots,d_n)$ among attributes and containment relationships of the entity type E_i, whose name is L_D.name and L_D.type = C_T.name. Each $t_i \in$ T.content originates a component d_i of D_k. If t_i is an element declaration, for example, then the corresponding d_i is an attribute or a containment relationship. If t_i is a compositor, then the corresponding d_i is a group of attributes and containment relationships. Additionally, when a disjunction constraint D_i is defined for a given entity E_i, then the set of disjunction constraints associated with the entity E_i must be updated in order to include the constraint D_i ($\underline{E_i.D} = \underline{E_i.D} \cup \{D_K\}$).

In the Computer Science Department schema there is only one choice group, which is defined in the content model of the professorTy complex type. To represent this choice group a disjunction constraint D_k({firstName,lastName},name))is defined between the name attribute and the group composed by firstName and lastName attributes of the professor entity type. Such constraint specifies that a given element professor contains either a name attribute or both firstName and lastName attributes.

- **Rule 6 - Conversion of key constraints**

A key definition K_D =(name,selector,field*)[4] originates one or more key attributes, which may belong either to the entity type E_i, whose name is K_D.selector, or to an entity type associated with E_i through a containment relationship (or a sequence of containment relationships). Each K_D.field specifies a key attribute. Since in the XML Schema language

[4] selector is the name of the element that defines the scope of the key and field is the name of an element or attribute which is restricted by the key definition.

more than one key definition may be associated with a single entity type, a key attribute has a property, called `key`, whose value is K_D`.name`.

Rule 6 defines how to map a key definition to its corresponding representation in the conceptual schema. In the `Computer Science Department` schema, there is only one key constraint which originates the definition of `number` as a key attribute of the `course` entity type, which means that the value of the `number` attribute must be unique and cannot be set to nil in the set of `course` elements.

- **Rule 7 - Conversion of keyref constraints**

A keyref definition F_D =(`name,key,selector,field*`)[5], originates a reference relationship R_j between the entity type E_i whose name is F_D`.selector` and the entity type E_i' whose name is K_D`.selector`, where K_D`.name` = F_D`.key`. To correctly represent a keyref definition extra information must be added to R_j. Besides the two participating entity types, information about the referencing and referenced attributes must be stored in R_j, i.e, R_j`(`E_i`,`E_i'`,keyrefAtt,keyAtt)`, where `keyrefAtt` is the list of referencing attributes and `keyAtt` is the list of referenced attributes. `keyAtt` is defined by the key definition K_D, where K_D`.name` = F_D`.key` and `keyrefAtt` is defined by the values of F_D`.field`. Additionally, when a reference relationship R_i is defined between the entity types E_1 and E_2, then the set of relationships associated with the entity E_1 must be updated in order to include the relationship R_i ($\underline{E_1.R} = \underline{E_1.R} \cup \{R_i\}$).

Rule 7 defines how to map keyref constraints to the conceptual schema. This is done creating a new reference relationship between the entity type that defines the scope of the keyref definition and the entity type that specifies the scope of the associated key constraint. In our example, there is a keyref definition, called `profCourse`, which specifies that the `courseNumber` attribute of the `professor` entity references the `courseNumKey` key. This constraint originates the following reference relationship: `professor_ref_course (professor,course,(courseNumber),(number))`.

4.3.3 Creating an X-Entity Schema

In the following, we present the conversion algorithm, which receives as input a pre-processed *XML Schema* and produces as output an X-Entity schema. The first task of the algorithm is the creation of the entity type definitions. All global element declarations and

[5] `key` is the name of referenced key, `selector` is the name of the element that defines the scope of the keyref and `field` is the name of an element or attribute which is restricted by the keyref definition.

complex type definitions are analyzed in order to map elements composed by other elements or attributes into entity types. The next task, consists of defining the attributes and relationships for the entity types identified earlier. To do this, each complex type definition is evaluated again and elements with base types and attribute declarations are mapped to attributes. Additionally, a new containment relationship is created whenever an element declaration with a complex type is found. The final tasks are the mapping of choice groups to disjunction constraints and the conversion of key and keyref definitions. Figure 4.14 presents the X-Entity schema for the Computer Science Department *XML Schema* presented in Figure 4.12.

```
Conversion(pre-processed XML Schema)
/* Creation of the entities */
For each global element declaration
  Apply Rule 1
For each complex type definition C_T
  For each element declaration L_D ∈ C_T.content
    If L_D.type is a complex type
    Then apply Rule 1
/* Creation of attributes, relationships and disjunction constraints*/
For each complex type definition C_T
  For each element declaration L_D ∈ C_T.content
    If L_D.type is a base type
    Then apply Rule 2
    Else apply Rule 4
  For each attribute declaration A_D ∈ C_T.content
    Apply Rule 3
  If there is a choice compositor C ∈ C_T.content
  Then apply Rule 5
/* Creation of key and keyref constraints */
For each key definition K_D
  Apply Rule 6
For each keyref definition F_D
  Apply Rule 7
```

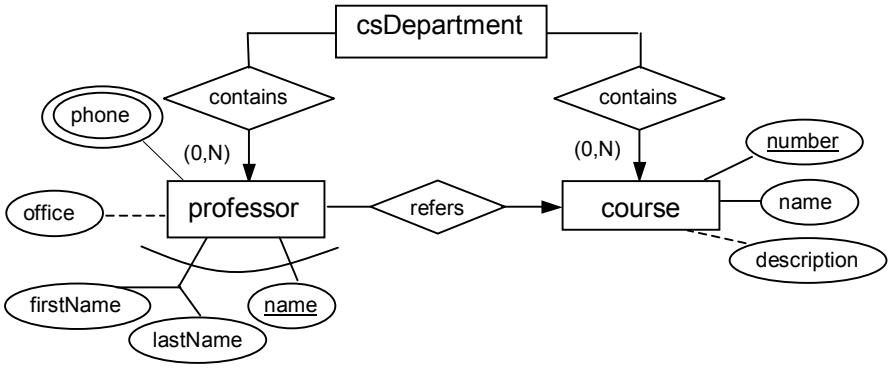

Figure 4.14 Computer Science Department X-Entity schema

We also propose the use of an XML document to specify X-Entity schemas. Figure 4.14 shows the XML specification for the `Computer Science Department` X-Entity schema. The XML specification has one `XENTITY_SCHEMA` element (the root element), which is composed by `ENTITY`, `CONTAINMENT_RELATIONSHIP` and `REFERENCE_RELATIONSHIP`

elements. An ENTITY element describes the attributes, relationships and disjunction constraints associated with an entity type through the ATTRIBUTE, RELATIONSHIP_NAME and CHOICE elements, respectively. A CONTAINMENT_ RELATIONSHIP element is composed by two elements: ELEMENT_ENTITY and SUBELEMENT_ENTITY, which represent the entity types participating in the relationship. A REFERENCE_RELATIONSHIP is also composed by two elements representing the participating entity types: REFERENCING_ENTITY and REFERENCED_ENTITY elements. Additionally, a REFERENCE_RELATIONSHIP has a KEY and a KEYREF elements, which specify the attributes involved in the reference relationship.

```
<XENTITY_SCHEMA name = "csDepartment Description">
  <ENTITY name="csDepartment">
    <RELATIONSHIP_NAME name="csDepartment_professor"/>
    <RELATIONSHIP_NAME name="csDepartment_course"/>
  </ENTITY>
  <ENTITY name="professor">
    <CHOICE>
      <ATTRIBUTE name="name" type="string" cardMin ="1" cardMax="1"/>
      <GROUP>
        <ATTRIBUTE name="firstName" cardMin ="1" cardMax="1"/>
        <ATTRIBUTE name="lastName" cardMin ="1" cardMax="1"/>
      </GROUP>
    </CHOICE>
    <ATTRIBUTE name="phone" type="string" cardMin ="1" cardMax="n"/>
    <ATTRIBUTE name="office" type="string" cardMin ="0" cardMax="1"/>
    <ATTRIBUTE name="level" type="string" cardMin ="1" cardMax="1"/>
    <ATTRIBUTE name="courseNumber" type="string" cardMin ="1" cardMax="n"/>
    <RELATIONSHIP_NAME name="professor_ref_course"/>
  </ENTITY>
  <ENTITY name="course">
    <ATTRIBUTE name="name" type="string" cardMin ="1" cardMax="1"/>
    <ATTRIBUTE name="number"type="string" cardMin ="1" cardMax="1" key="key1"/>
    <ATTRIBUTE name="description" type="string" cardMin ="0" cardMax="1"/>
    <KEY name="key1"/>
  </ENTITY>
  <CONTAINMENT_RELATIONSHIP name="csDepartment_professor" cardMin ="1" cardMax="n">
    <ELEMENT_ENTITY name="csDepartment"/>
    <SUBELEMENT_ENTITY name="professor"/>
  </CONTAINMENT_RELATIONSHIP>
  <CONTAINMENT_RELATIONSHIP name="csDepartment_course" cardMin ="1" cardMax="n">
    <ELEMENT_ENTITY name="csDepartment"/>
    <SUBELEMENT_ENTITY name="course"/>
  </CONTAINMENT_RELATIONSHIP>
  <REFERENCE_RELATIONSHIP name="professor_ref_course">
    <REFERENCING_ENTITY name="professor"/>
    <REFERENCED_ENTITY name="course"/>
    <KEY name="key1">
      <ATTRIBUTE_NAME name="number"/>
    </KEY>
    <KEYREF name="keyref1">
      <ATTRIBUTE_NAME name="courseNumber"/>
    </KEYREF>
  </REFERENCE_RELATIONSHIP>
</XENTITY_SCHEMA>
```

Figure 4.15 – XML specification for Computer Science Department X-Entity schema

4.4 Concluding remarks

This chapter presented X-Entity, a conceptual data model for XML schemas. The X-Entity model describes the hierarchical structure of XML schemas using a flat representation that highlights entities and the relationships among them. Such representation provides a cleaner description for XML schemas hiding implementation details and focusing on semantically relevant concepts. We also presented the process of converting an *XML Schema* to an X-Entity schema. The conversion process is based on a set of rules that consider element declarations and type definitions of an *XML Schema* and generates the corresponding conceptual elements.

The X-Entity model extends the ER model so that one can explicitly represent important features of XML schemas, including: element and subelement relationships, occurrence constraints of elements and attributes, choice groups and references between elements. Other issues were not considered in our work, including: hierarchy of elements and attributes, cardinality of group of elements, elements with mixed content and order of elements imposed by a sequence compositor. However, our model can be easily extended with additional features and new rules can be developed for the conversion process.

One important advantage of using the X-Entity model is that each entity can be seen and manipulated as an individual concept even if it belongs to a nested structured. Instead of nested entities, each entity has a set of relationships that represents its association with other entities. This kind of representation facilitates mediation queries generation and maintenance.

The X-Entity model represents an XML Schema as a set of entity types and relationship types. This representation helps the identification of real world concepts, which will be used during user queries definition and therefore must be associated with a mediation query. Moreover, the X-Entity model makes distinction between entity types and attributes. During, mediation queries generation entity types and attributes are manipulated in different ways. In an XML schema there is no clear distinction when an information must be represented as an element or an attribute.

Some recent work have discussed the conceptual data modeling of XML schemas using the ER model. In [Psaila 2000] is proposed the ERX conceptual model, an evolution of the classical ER model which provides specific features suitable to model large collections of XML documents. More precisely, ERX extends the ER model to allow the representation of style sheets and a collection of documents conforming to a DTD. In [Passi et al. 2002] is proposed an object-oriented data model, called XSDM (XML Schema Data Model), to represent *XML Schemas*. XSDM was proposed as a model for XML Schema integration, i.e., during the first

step of the schema integration *XML Schemas* are translated into XSDM notation. [Psaila 2000] and [Passi et al. 2002] present conceptual models but they do not discuss XML schema conversion rules, which makes hard using the proposed conceptual models.

The work presented in [Mani et al. 2001] proposes a new notation, called XGrammar, to formalize the most important features from the proposed XML schema languages. Mani et. al. also extends the ER model with additional features (order in a binary relationship and element-subelement relationship) to support the XML model. [Mani et al. 2001] presents conversion rules from XGrammar to ER model extensions, but to use that approach it will be necessary to define an additional mapping from an XML schema to XGrammar. In [Mello et al. 2001] it is described a semi-automatic process for converting a DTD to a conceptual schema in a canonical conceptual model. A broadly discussion of the conversion from a DTD to its corresponding conceptual schema is presented, but they do not make distinction between entity types and attributes.

Chapter 5

Generating Mediation Queries for XML-based Mediators

5.1 Introduction

In this chapter we describe the process of generating mediation queries for XML-based mediators. As mentioned earlier, the process of mediation queries generation is based on the approach proposed in [Kedad et al. 1999], which provides a support to discover queries over a set of heterogeneous sources in a GAV context. Such approach defines a solution space which provides the set of potential queries that corresponds to a given relational view. As we consider XML as our common data model we had to adapt this approach to the context of XML data.

In the GAV approach, a mediation query describes how to compute an element of the mediation schema over the data sources. In our work, the mediation schema is represented by an X-Entity schema, which consists of entities and relationships among them. Therefore, the process of mediation queries generation consists in discovering a computing expression for each entity in the mediation schema.

More formally, we can say that defining a mediation query for a mediation entity E_m consists in decomposing E_m into n entities $E_{p1},...,E_{pn}$ such that $E_m = E_{p1}\theta_1 E_{p2}\theta_2...\theta_{n-1}E_{pn}$, where θ_i is a binary operator and each entity E_{pi} is derived using an expression $Exp(E_i)$ over a single source entity E_i. Entities E_{pi} are called relevant source entities to compute E_m. The process of discovering queries, which defines a mediation entity, can be summarized in the following.

– *Selection of relevant source entities which potentially allow to compute a given mediation entity*;

– *Identification of possible operators to apply between different and relevant source entities*; *and*

– *Generation of all possible mediation queries from the selected source entities and operators*.

The process of mediation queries generation is widely based on the existence of metadata describing individual sources and the mediation schema, and on correspondence assertions [Spaccapietra et al. 1994] specifying relationships between concepts of different schemas. We propose different types of correspondence assertions to formally describe the relationships among concepts of X-Entity schemas.

The approach adopted for mediation queries generation provides a formalism to represent such queries, which is based on the concept of operation graphs [Kedad et al. 1999]. Relevant source entities are the nodes of the operation graph while the edges are labeled with the operators to be applied among the source entities. One of the problems faced at the beginning of our work, concerning the evolution of mediation queries, was the absence of formalism to be used in their representation. Using this approach, the problem of mediation queries evolution becomes the problem of maintaining the operation graphs representing the mediation queries.

This chapter is organized as follows. Section 5.2 describes in more details the notation and the metadata used. The following sections introduce each task of the process of mediation queries generation: section 5.3 describes how to select the relevant source entities to compute a given mediation entity, section 5.4 discusses the identification of operators to apply between source entities and the generation of the computing expressions. Section 5.5 describes how to compute mediation entities from mediation queries. Section 5.6 shows some examples of how to compute user queries from mediation entities. Finally, section 5.7 summarizes the chapter.

5.2 Terminology

We use the following notation to denote schemas, entities and attributes.

– $E_i.A$: denotes the set of attributes of the entity type E_i.

– $E_i.R$: denotes the set of relationship types of the entity type E_i.

– $S.E$: denotes the set of entity types of the schema S.

– $S.R$: denotes the set of relationship types of the schema S.

We use correspondence assertions to represent correspondences between entities and attributes of distinct sources. Correspondence Assertions, as defined in [Spaccapietra et al. 1994, Vidal et al. 2001], are special types of integrity constraints used to assert that the

semantics of some components in a schema is somehow related to the semantics of some components in another schema. In our approach, correspondence assertions are used just to solve problems concerning semantic heterogeneity between X-Entity elements, i.e., they are used to represent correspondences between X-Entity elements representing the same real world concept and having the same semantics. The correspondence assertions are defined between elements having the same structure, i.e., we identify correspondences just between entity types and correspondences between attributes. We identify the following types of correspondence assertions to formally describe the relationships between the concepts of X-Entity schemas:

- *Entity correspondence assertions*: specify the relationship between entity types of distinct schemas.

  Definition 5.1 (Entity correspondence assertion): Let E_1 and E_2 be entity types (a source entity type or a mediation entity type). The correspondence assertion $E_1 \cong^6 E_2$ specifies that E_1 and E_2 are semantically equivalent, i.e, they describe the same real world concept and they have the same semantics.

  Consider, for example, the X-Entity schemas[7] presented in Figure 5.1 and Figure 5.2. After the matching of these two schemas the following entity assertions are obtained:

  – $movie_1 \cong movie_2$: specifies that the $movie_1$ and $movie_2$ entity types are semantically equivalent.

  – $director_1 \cong director_2$: specifies that the $director_1$ and $director_2$ entity types are semantically equivalent.

  Besides the intensional correspondences, it is important to define the relationship between the extensions (set of instances) of semantically equivalent entities. An instance of an entity is a particular element in an XML document source. The correspondences between the extensions of semantically equivalent enitites are determined through correspondence assertions specifying which of the usual set relationships holds: equivalence (≡), inclusion(⊂) and intersection(∩).

```
Schema of data source S1 =
({movie1({title1,duration1,genre1},movie1_actor1, movie1_director1}),
  actor1({name1, nationality1},{})
  director1({name1, nationality1},{})},
 {movie1_actor1(movie1,actor1,(1,N)),
  movie1_director1(movie1,director1, (1,N))})
```

[6] In [Rahm et al. 2001] the symbol "≅" is used to represent that a certain element of a schema S_1 corresponds to a certain element of a schema S_2.

[7] To facilitate the understanding of the correspondence assertions we omit the root element from the X-Entity schemas used in this section.

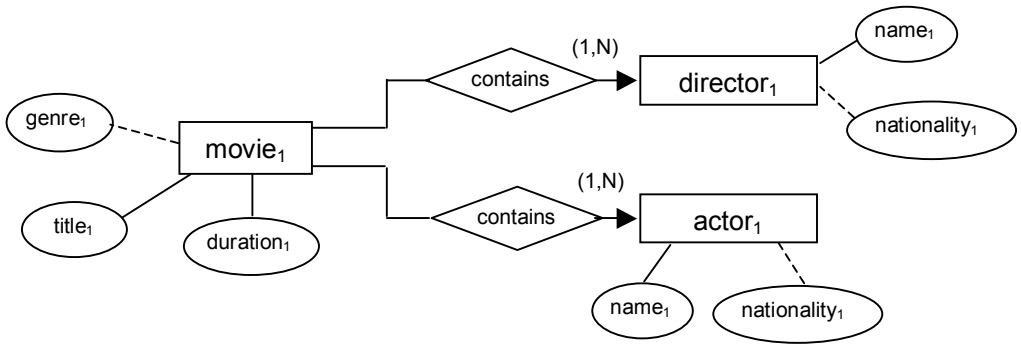

Figure 5.1 – Movie schema S_1

Schema of data source S_2 =
({$movie_2$({$title_2$, $genre_2$, $actor_2$},{$movie_2$_$director_2$}),
  $director_2$({$name_3$, $nationality_3$},{})},
 {$movie_2$_$director_2$($movie_2$,$director_2$,(1,N))})

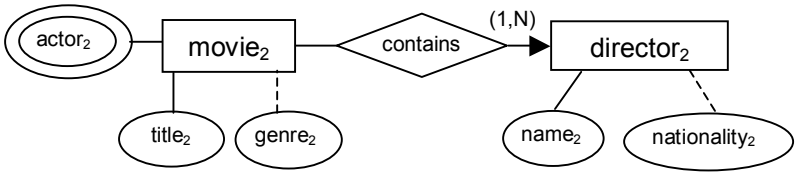

Figure 5.2 – Movie schema S_2

The expression $C(E_i)$ denotes the collection of XML elements $\{e_1,\ldots,e_n\}$ which are instances of a given entity type E_i (E_i represents the element declaration that defines the type of each element e_j). For example, $C(movie_1)$ denotes the collection of $movie_1$ instances. When an element e_j is composed by subelements $\{e_{j1},\ldots,e_{jm}\}$, then the expression $C(E'(e_j))$ is used to denote the set of elements e_{jk} that are instances of the entity type E'. Consider for example, the element e_1 represented below.

e_1 = <$movie_1$>
  <$title_1$> Gangs of New York </$title_1$>
  <$director_1$> Martin Scorsese </$director_1$>
  <$theater_1$>
    <$name_1$> Art Iguatemi </$name_1$>
    <$time_1$> 17h30 </$time_1$>
    <$time_1$> 20h40 </$time_1$>
  </$theater_1$>
  <$theater_1$>
    <$name_1$> North Shopping </$name_1$>
    <$time_1$> 17h10 </$time_1$>
    <$time_1$> 20h30 </$time_1$>
  </$theater_1$>
  <$theater_1$>
    <$name_1$> Del Paseo </$name_1$>
    <$time_1$> 17h30 </$time_1$>
    <$time_1$> 20h30 </$time_1$>
  </$theater_1$>
</$movie_1$>

$C(theater(e_1)):\{e_{11}, e_{12}, e_{13}\}$, where:

```
e11 = <theater>
          <name> Art Iguatemi </name>
          <time> 17h30 </time>
          <time> 20h40 </time>
      </theater>
e12 = <theater>
          <name> North Shopping </name>
          <time> 17h10 </time>
          <time> 20h30 </time>
      </theater>
e13 = <theater>
          <name> Del Paseo </name>
          <time> 17h30 </time>
          <time> 20h30 </time>
      </theater>
```

Considering the X-Entity schemas presented in Figure 5.1 and Figure 5.2, the extensional correspondence between the entities $movie_1$ and $movie_2$ is represented by the following assertion $movie_1 \cap movie_2 \neq \emptyset$, which specifies that there is an intersection between the collection of $movie_1$ instances of data source S_1 and the collection of $movie_2$ instances of data source S_2. The extensional correspondence between the entities $director_1$ and $director_2$ is represented by the following assertion $director_1 \cap director_2 \neq \emptyset$, which specifies that there is an intersection between the collection of $director_1$ instances of data source S_1 and the collection of $director_2$ instances of data source S_2.

For each pair of semantically equivalent entity types we specify the correspondences between their subentities and attributes using subentity correspondence assertions and attribute correspondence assertions. A subentity of an entity type E_i is an entity type E_i' which is associated with E_i through a containment relationship $R_j(E_i, E_i', (min, max))$.

- *Subentity correspondence assertions*: specify the correspondence among subentities of semantically equivalent entity types.

Definition 5.2 (Subentity correspondence assertion): Let $R_1(E_1, E', (min, max))$ and $R_2(E_2, E'', (min, max))$ be two containment relationships, such that $E_1 \cong E_2$ and $E' \cong E''$. The correspondence assertion $E_1.R_1.E' \cong E_2.R_2.E''$ specifies that the subentity E' of the entity type E_1 is semantically equivalent to the subentity E'' of the entity type E_2.

The following subentity assertion was obtained as a result of the matching of the schemas S_1 and S_2 presented in Figures 5.1 and 5.2.

– $movie_1.movie_1\_director_1.director_1 \cong movie_2.movie_2\_director_2.director_2$

- *Attribute correspondence assertions*: specify the correspondence among common attributes of semantically equivalent entities.

  Definition 5.3 (Attribute correspondence assertion): Let A_1 be an attribute[8] of the entity type E_1 and A_2 an attribute of the entity type E_2, such that $E_1 \cong E_2$. The correspondence assertion $E_1.A_1 \cong E_2.A_2$ specifies that the attributes A_1 and A_2 are semantically equivalent (correspond to the same concept in the real world).

  The following attribute assertions were obtained as a result of the matching of the schemas S_1 and S_2 presented in Figures 5.1 and 5.2 respectively.

  – $movie_1.title_1 \cong movie_2.title_2$
  – $movie_1.genre_1 \cong movie_2.genre_2$
  – $director_1.name_1 \cong director_2.name_2$
  – $director_1.nationality_1 \cong director_2.nationality_2$

  To compute a given mediation entity it may be necessary to access a few properties (attributes and containment relationships) of the entity in one data source, and other properties in another data source. Therefore, the mediation system has to know how to find in one data source the element corresponding to a given element in another data source. To specify this kind of correspondence we use a special type of attribute assertion, which is defined below.

  Definition 5.4 (Mapping attribute correspondence assertion): Let $(A_{11},\ldots,A_{1n})$ be monovalued attributes of the entity type E_1 and $(A_{21},\ldots,A_{2n})$ monovalued attributes of the entity type E_2, such that $E_1 \cong E_2$ and $\forall A_{1i} \in (A_{11},\ldots,A_{1n}) \wedge \forall A_{2i} \in (A_{21},\ldots,A_{2n}), 1 \leq i \leq n, A_{1i} \cong A_{2i}$. The correspondence assertion $E_1.(A_{11},\ldots,A_{1n}) \cong E_2.(A_{21},\ldots,A_{2n})$ specifies the mapping between the corresponding instances of E_1 and E_2. If two instances e_1 and e_2 of the entities E_1 and E_2, respectively, have the same values for each pair of semantically equivalent attributes A_{1i} and A_{2i}, $1 \leq i \leq n$, then e_1 and e_2 are corresponding instances, i.e., they represent the same object in the real world. The attributes $(A_{11},\ldots,A_{1n})$ and $(A_{21},\ldots,A_{2n})$ are called mapping attributes between E_1 and E_2.

  In our example, the mapping between instances of semantically equivalent entities is defined by the following attribute assertions:

[8] In this work, we consider just monovalued attributes. However, our approach may be extended to consider multivalued attributes. If the attributes have different domains or different structures conversion functions and mapping function must be used.

– $movie_1.(title_1) \cong movie_2.(title_2)$: specifies that when an instance m_1 of $movie_1$ has a value "t" for the attribute $title_1$ and an instance m_2 of $movie_2$ has the same value "t" for the attribute $title_2$, then m_1 and m_2 are corresponding instances ($m_1 \equiv m_2$).

– $director_1.(name_1) \cong director_2.(name_2)$: specifies that when an instance d_1 of $director_1$ has a value "d" for the attribute $name_1$and an instance d_2 of $director_2$ has the same value "d" for the attribute $name_2$, then d_1 and d_2 are corresponding instances ($d_1 \equiv d_2$).

Mapping attributes are very useful during the extension correspondence assertions identification. To identify the relationships between the entity types extensions the mapping attributes values of semantically equivalent entity types must be compared. To help such identification, heuristics must also be defined.

- *Path correspondence assertions*: specify special types of correspondences between attributes and subentities of semantically equivalent entity types having different structures. To define a path assertion, we first have to define the concepts: link and path.

  Definition 5.5 (Link): Let X_1 and X_2 be elements of an X-Entity schema (an element can be an entity type, a containment relationship type or an attribute), $X_1.X_2$ is a link if:

  i) X_2 is an attribute of the entity type X_1, or

  ii) X_1 is an entity type participating in the relationship type X_2 (or vice-versa).

  Definition 5.6 (Path): If $X_1 \ldots . X_n$ are elements of a schema, such that $\forall X_i$, $1 \leq i \leq n$, $X_i.X_{i+1}$ is a link, then $X_1.X_2. \ldots .X_n$ is a path from X_1.

  Definition 5.7 (Inverse path): If there is a path from X_1 denoted by $X_1.X_2. \ldots .X_n$, such that X_n is an entity type, then there is a path denoted by $(X_1.X_2. \ldots .X_n)^{-1}$, which is called the inverse path of $X_1.X_2. \ldots .X_n$.

  Definition 5.8 (Path correspondence assertion): Let P_1 and P_2 be two paths:

  Case 1: $P_1 = X_1.X_2. \ldots .X_n$ and $P_2 = Y_1.Y_2. \ldots .Y_n$, where $X_1 \cong Y_1$. The correspondence assertion $P_1 \cong P_2$ specifies that the entity types X_n and Y_n are semantically equivalent.

  Case 2: $P_1 = X_1.X_2. \ldots .X_n.A_k$ and $P_2 = Y_1.Y_2. \ldots .Y_n.A_k'$, where $X_1 \cong Y_1$. The correspondence assertion $P_1 \cong P_2$ specifies that the attribute A_k of the entity type X_n and the attribute A_k' of the entity type Y_n are semantically equivalent.

  Case 3: $P_1 = X_1.X_2. \ldots .X_n$ and $P_2 = (Y_1.Y_2. \ldots .Y_n)^{-1}$, where $X_1 \cong Y_n$. The correspondence

assertion $P_1 \cong P_2$ specifies that the types X_n and Y_1 are semantically equivalent.

Case 4: $P_1 = X_1.X_2.\ldots.X_n.A_k$ and $P_2 = (Y_1.Y_2.\ldots.Y_n)^{-1}.A_k'$, where $X_1 \cong Y_n$. The correspondence assertion $P_1 \cong P_2$ specifies that the attribute A_k of the entity type X_n and the attribute A_k' of the entity type Y_1 are semantically equivalent.

In our example, after the matching of the schemas S_1 and S_2 the following path assertion is obtained:

– $movie_2.actor_2 \cong movie_1.movie\_actor_1.actor_1.name_1$: specifies that the attribute $actor_2$ of the entity $movie_2$ and the attribute $name_1$ of the entity $actor_1$ are semantically equivalent.

In the schema S_2 information about actors of a movie is represented by the actor attribute while in the schema S_1 the same information is represented by the entity actor.

Definition 5.9 (Atributte derivation path): A path assertion $E_i.A_k \cong P.A_k'$, where A_k and A_k' are attributes and $P = Y_1.Y_2.\ldots.Y_n$ or $P = (Y_1.Y_2.\ldots.Y_n)^{-1}$, specifies that $P.A_k'$ is a derivation path of the attribute A_k.

In our example, the correspondence assertion $movie_2.actor_2 \cong movie_1.actor_1.name_1$, specifies that $movie_1.actor_1.name_1$ is a derivation path of the $actor_2$ attribute.

Definition 5.10 (Entity derivation path): A path assertion $E_i.R_j.E_i' \cong P$, where R_j is a relationship type, E_i and E_i' are entity types and $P = Y_1.Y_2.\ldots.E_i''$ or $P = (E_i''.Y_2.\ldots.Y_n)^{-1}$, specifies that P is a derivation path of the entity E_i'.

Suppose the X-Entity schema S_3 presented in Figure 5.3, which represents the $movie_3$ entity as a subentity of the $director_3$ entity. Matching the schema S_1 with the schema S_3 we obtain the correspondence assertions presented in Table 5.1. Observe that, in this case, the subentity $director_1$ of $movie_1$ corresponds to a derivation path, which is an inverse path from $director_3$:

- $(movie_1.movie_1\_director_1.director_1 \cong (director_3.director_3\_movie_3.movie_3)^{-1})$

```
Schema S3 =
({director3({name3,nationality3},{director3_movie3}),
  movie3({title3,year3},{}),
 {director3_movie3(director3,movie3,(1,N))})
```

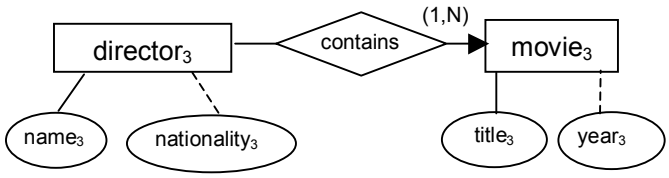

Figure 5.3 – Movie schema S_3

Table 5.1- Correspondence assertions between the schemas S_1 and S_3

Correspondence assertions between schemas S_1 and S_3
Entity correspondence assertions:
$movie_1 \cong movie_3$
$movie_1 \cap movie_3 \neq \emptyset$
$director_1 \cong director_3$
$director_1 \cap director_3 \neq \emptyset$
Attribute correspondence assertions:
$movie_1.title_1 \cong movie_3.title_3$
$director_1.name_1 \cong diector_3.name_3$
$director_1.nationality_1 \cong diector_3.nationality_3$
Mapping attribute correspondence assertions:
$movie_1.(title_1) \cong movie_3.(title_3)$
$director_1.(name_1) \cong director_3.(name_3)$
Path correspondence assertions:
$movie_1.movie_1\_director_1.director_1 \cong$
$(director_3.director_3\_movie_3.movie_3)^{-1}$

Table 5.2 summarizes all correspondence assertions obtained as a result of the matching of the two schemas S_1 and S_2.

Table 5.2- Correspondence assertions between the schemas S_1 and S_2

Correspondence assertions between schemas S_1 and S_2
Entity correspondence assertions:
$movie_1 \cong movie_2$
$movie_1 \cap movie_2 \neq \emptyset$
$director_1 \cong director_2$
$director_1 \cap director_2 \neq \emptyset$
Subentity correspondence assertions:
$movie_1.director_1 \cong movie_2.director_2$
Attribute correspondence assertions:
$movie_1.genre_1 \cong movie_2.genre_2$
$movie_1.title_1 \cong movie_2.title_2$
$director_1.name_1 \cong diector_2.name_2$
$director_1.nationality_1 \cong diector_2.nationality_2$
Mapping attribute correspondence assertions:
$movie_1.(title_1) \cong movie_2.(title_2)$
$director_1.(name_1) \cong director_2.(name_2)$
Path correspondence assertions:
$movie_2.actor_2 \cong$
$movie_1.movie_1\_actor_1.actor_1.name_1$

5.3 Determination of relevant source entities

The first task for generating a mediation query for a given mediation entity E_m is the determination of the source entities which are relevant for its computation. Intuitively, a source entity E_i is relevant to the computation of the mediation entity E_m if E_i and E_m are semantically equivalent, i.e., E_i and E_m represent the same real world concept. The links between E_m and its relevant source entities are represented by the following mapping views.

- *Basic mapping views (m-views)*

A mapping view specifies how to compute attributes and subentities of a mediation entity E_m from a source entity E_i. A mapping view $V(\{X_1,\ldots,X_n\},\{Y_1,\ldots,Y_m\})$ is a special type of entity type where X_i is an attribute or an attribute derivation path and Y_i is a relationship or an entity derivation path.

Let $E_m(\{A_{m1},\ldots,A_{mn}\},\{R_{m1},\ldots,R_{mk}\})$[9] be a mediation entity with n attributes and k containment relationships, and S a set of data sources containing a particular entity $E_i(\{A_{i1},\ldots,A_{ip}\},\{R_{i1},\ldots,R_{iq}\})$, such that $E_m \cong E_i$. The mapping view $V(X,Y)$ over E_i is defined as described below:

$\forall A_m \in \underline{E_m.A}$:

if $\exists A_k \in \underline{E_i.A} \mid E_m.A_m \cong E_i.A_k$ *then* $X = X \cup \{A_k\}$

if $\exists E_i.\ldots.E_k.A_k \mid E_m.A_m \cong E_i.\ldots.E_k.A_k$ *then* $X = X \cup \{E_i.\ldots.E_k.A_k\}$

if $\exists E_k.\ldots.E_{ij} \mid E_m.A_m \cong (E_k.\ldots.E_i)^{-1}.A_k$ *then* $X = X \cup \{(E_k.\ldots.E_i)^{-1}.A_k\}$

$\forall R_{med}(E_m,E_m') \in \underline{S_m.R}$

if $\exists R_k(E_i, E') \in \underline{S_i.R} \mid E_m.R_m.E_m' \cong E_i.R_k.E'$ *then* $Y = Y \cup \{R_k\}$

if $\exists E_i.\ldots.E' \mid E_m.R_m.E_m' \cong E_i.\ldots.E'$ *then* $Y = Y \cup \{E_i.\ldots.E'\}$

if $\exists E'.\ldots.E_i \mid E_m.R_m.E_m' \cong (E'.\ldots.E_i)^{-1}$ *then* $Y = Y \cup \{(E'.\ldots.E_i)^{-1}\}$

The correspondence assertions between the mediation entity E_m and the source entities are identified during the determination of relevant source entities. This task is achieved by the *Schema Matcher* (c.f. Chapter 3).

Consider, for example, the mediation schema presented in Figure 5.4, which integrates information from the source schemas presented in Figures 5.1, 5.2 and 5.3.

[9] We ommit the disjunction constraints and reference relationships because they are not been considered in the process of determining the relevant source entities. We also ommit the cardinality of the containment relationships when they are not seen as relevant.

```
Mediation Schema S_med =
({movie_m({title_m, genre_m, year_m,director_m}, {movie_m_actor_m}),
  actor_m({name_m, nationality_m},{})},
 {movie_m_actor_m(movie_m,actor_m,(1,N))})
```

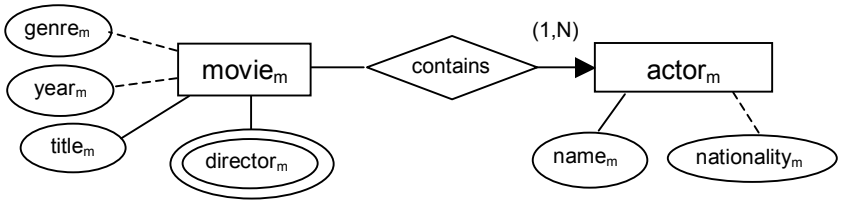

Figure 5.4 – Mediation schema of data source S_{med}

The set of m-views V_{Movie1}, V_{Movie2} and V_{Movie3} associated with the mediation entity $movie_m$ as well as the set of m-views V_{Actor1} and V_{Actor3} associated with the mediation entity $actor_m$ are presented below. The set of all m-views associated with E_m over S is denoted M_{Em}.

A mapping view has only attributes, relationships and derivation paths which are relevant for the computation of its associated mediation entity. For example, the source entity $movie_1$ has a $duration_1$ attribute, which does not belong to the mapping view V_{Movie1} because it is not relevant for the computation of $movie_m$.

```
Mapping views of movie_m:
  V_Movie1({title_1, genre_1, movie_1.movie_1_director_1.director_1.name_1},
        {movie_1_actor_1})
  V_Movie2({title_2, genre_2, movie_2.movie_2_director_2.director_2.name_2}, {})
  V_Movie3({title_3,(director_3.director_3_movie_3,movie_3)^-1.name_3,year_3}, {})
Mapping views of actor_m:
  V_Actor1 ({name_1, nationality_1},{})
```

As we may observe, to compute a given mediation entity it may be necessary to access a few properties (attributes and containment relationships) of the entity in one data source, and other properties in another data source. To do this, we have to specify the mapping attributes between the source entities. These attributes are used to determine how to find in one data source the element corresponding to a given element in another data source.

- *Extended mapping views (xm-views)*

It may occur that the m-views are not sufficient to derive the query of a given mediation entity. Consider the example of a mediation entity $movie_m(\{title_m, director_m, genre_m\},\{\})$ and the source entities $movie_1(\{title_1, director_1, year_1\},\{\})$ and $movie_2(\{title_2, genre_2, year_2\},\{\})$, such that $movie_1.(title_1,year_1) \cong movie_2.(title_2,year_2)$. The m-views associated to $movie_m$ are $V_{Movie1}(\{title_1, director_1\},\{\})$ and $V_{Movie2}(\{title_2,genre_2\},\{\})$. Because the attributes $year_1$ and

$year_2$ do not belong to the mapping views V_{Movie1} and V_{Movie2}, there is no way to identify corresponding instances and, therefore, it is not possible to obtain instances having all attributes required in the mediation entity $movie_m$. In this case, to perform the merge between corresponding instances, we have to add to these m-views additional attributes. The attributes $year_1$ and $year_2$ are candidate attributes for such purpose; they allow detecting when two instances e_1 and e_2 of the mapping views V_{Movie1} and V_{Movie2} are corresponding instances and could be integrated. So, it will be possible to obtain integrated instances that have all the attributes specified in the mediation entity $movie_m$. The new m-views are: $V_{Movie1}(\{title_1, director_1, year_1\},\{\})$ and $V_{Movie2}(\{title_2, genre_2, year_2\}, \{\})$.These are called extended mapping views (xm-views).

An xm-view between a mediation entity E_m and a source entity E_i is defined as a m-view augmented with all attributes of E_i which appear as mapping attributes with other relevant source entities of E_m. In the remainder of this work, we use the term m-view to denote both mapping views and extended mapping views.

5.4 Computation paths associated with a mediation entity

The second task in the process of generating a mediation query for a given mediation entity E_m is the determination of the computation paths of E_m. The set of relevant source entities for a mediation entity E_m is composed by the set of m-views denoted M_{Em}. Searching computation paths associated with the mediation entity E_m starts by finding the set of operators which can be applied to each pair of m-views. These operators are represented by an operation graph, which will enable to characterize the possible computation paths for the mediation entity E_m.

In this section, we first introduce the operator definitions. Next, we present how to define an operation graph and how to determine the computation paths associated with a mediation entity.

5.4.1 The operators

The main purpose of the mapping operators is to combine mapping views in order to provide a means of computing a given mediation entity. The primary function of these operators is to perform the integration of corresponding instances. This task is called object fusion [Papakonstantinou et al. 1996] and involves grouping information together about the same real-world entity. When two instances e_1 and e_2 have the same values for their mapping attributes (e_1 and e_2 are corresponding instances), only one instance representing the

integration of e_1 and e_2 is inserted in the result collection. The structure of the integrated instance is determined by the mediation entity structure.

We can define the three operations mapping union, mapping intersection, and mapping difference on two entity types E_1 and E_2 as follows:

- $\cup_p$ (*mapping union*): the result of this operation, denoted by $E_1 \cup_p E_2$ is a collection that includes all instances that are either in $C(E_1)$ or $C(E_2)$ or in both $C(E_1)$ and $C(E_2)$.
- $\cap_p$ (*mapping intersection*): the result of this operation, denoted by $E_1 \cap_p E_2$ is a collection that includes all instances that are in both $C(E_1)$ and $C(E_2)$.
- $-_p$ (*mapping difference*): the result of this operation, denoted by $E_1 -_p E_2$ is a collection that includes all instances that are in $C(E_1)$ but not in $C(E_2)$.

When a mapping operator is applied over two entity types E_1 and E_2, the instances from $C(E_1)$ and $C(E_2)$ are joined together on the values of the mapping attributes between E_1 and E_2. We can merge the contents of two semantically equivalent instances, as described below:

  - All attributes and subelements[10] with atomic values of the source entities are merged together in the result instance. Redundancies among them must be eliminated based on their atomic values.
  - Subelements with complex structures must be repeated in the result instance. Subelements which are instances of semantically equivalent entity types must be grouped together in order to be integrated later using the corresponding mediation query.

The only restriction on the usage of the set operators between mapping views is that a mapping operator may be applied only when the mapping views have a common identifier which is defined through the mapping attributes. The mapping operator to be applied between two mapping views is determined based on the correspondence assertions that specify the relationship between the extensions of the source entities, as defined in Table 5.3. We use correspondence assertions in order to reduce the number of possible operators to be applied between two mapping views. Using correspondence assertions we may identify if the application of a given operator results in an empty collection, for example. When there is no intersection between the extensions of two entity types, then the only operator to be applied to merge their instances is the $\cup_p$. If we use other mapping operator then the result is an empty collection.

[10]If the atomic values have different cardinalities, conversion functions and concatenation functions, for example, may be used to make the conversion between them.

Table 5.3 – Mapping operators

Correspondence assertion	Operator
$E_1 \equiv E_2$	——
$E_1 \subset E_2$	$E_2 -_p E_1$, $E_1 \cup_p E_2$
$E_1 \cap E_2 \neq \varnothing$	$E_1 \cup_p E_2$, $E_1 \cap_p E_2$, $E_1 -_p E_2$, $E_2 -_p E_1$
$E_1 \cap E_2 = \varnothing$	$E_1 \cup_p E_2$

5.4.2 Operation graph

The set of candidate operations combining pairs of m-views is represented by an operation graph. In this graph, a node is associated with each m-view, and an edge between two nodes represents a mapping operator. Generally, these two entities belong to different data sources. Formally, we define an operation graph noted $G(M_{Em}, O_{Em})$ as follows:

- M_{Em} is the set of nodes of the graph G, representing the set of m-views associated with a mediation entity E_m.
- O_{Em} is the set of edges of the graph, labeled with one of the following operators: mapping union $(\cup_p)$, mapping intersection $(\cap_p)$ and mapping difference $(-_p)$.

The number of edges between two m-views in the operation graph will be equal to the number of possible operations between them. The construction of an operation graph $G(M_{Em}, O_{Em})$ associated with a mediation entity E_m can be easily obtained by applying to each pair of m-views the possible operators as defined in the earlier section. When the operation is a non-commutative one, the edge is directed according to the operation.

Figure 5.5 presents an example of an operation graph that describes the possible operators to combine the mapping views associated with the mediation entity $movie_m$ presented in the example of section 5.3. The correspondence assertions used to determine the operators are presented below.

$$
\begin{aligned}
&\text{Mediation Schema } S_{med} = \\
&\{(movie_m(\{title_m, genre_m, director_m, year_m\}, \{movie_m\_actor_m\}), \\
&\quad actor_m(\{name_m, nationality_m\}, \{\})\}, \\
&\quad \{movie_m\_actor_m(movie_m, actor_m)\}) \\
&\text{Mapping views of } movie_m: \\
&\quad V_{Movie1}(\{title_1, genre_1, movie_1.movie_1\_director_1.director_1.name_1\}, \\
&\qquad \{movie_1\_actor_1\}) \\
&\quad V_{Movie2}(\{title_2, genre_2, movie_2.movie_2\_director_2.director_2.name_2\}, \{\}) \\
&\quad V_{Movie3}(\{title_3, (director_3.director_3\_movie_3, movie_3)^{-1}.name_3, year_3\}, \{\})
\end{aligned}
$$

Extensional Correspondence assertions:

$movie_1 \cap movie_2 = \varnothing$

$movie_1 \cap movie_3 = \varnothing$

$movie_3 \subset movie_2$

The nodes of the operation graph, presented in Figure 5.5, represent the mapping views V_{Movie1}, V_{Movie2} and V_{Movie3}. The edges beween these mapping views represent the possible operators (mapping union and mapping difference), which were determined based on the extensional correspondence assertion presented above.

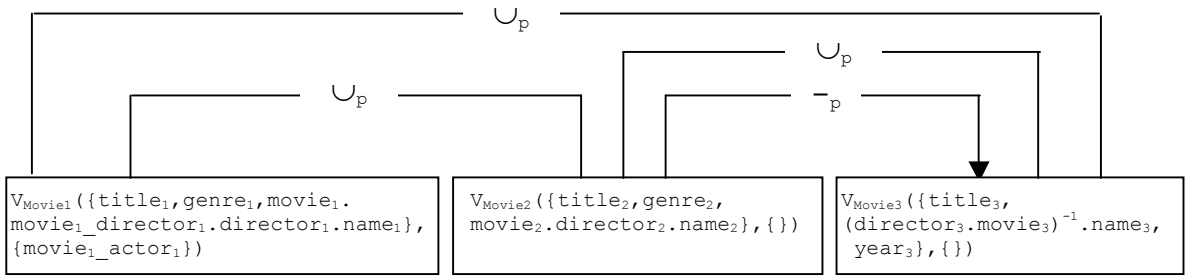

Figure 5.5 – Example of an operation graph

5.4.3 Computation path

The determination of the computation paths associated with a mediation entity consists of searching the set of operations, which enables to compute this entity. This search is done using the operation graph $G(M_{Em}, O_{Em})$. Formally, a computation path C_{Em} is defined by one of the following rules:

– An m-view, which involves all the attributes/subentities of the mediation entity E_m, is a computation path.

– Any connected subgraph of the operation graph $G(M_{Em}, O_{Em})$, which involves all the attributes and subentities of E_m is a computation path.

A computation path includes the set of operations which are necessary to compute all the attributes and subentities of the mediation entity. The general form of a computation path is the following:

$C = \{\theta[V_1, V_2], \theta[V_2, V_3], \ldots, \theta[V_k, V_{k+1}], \ldots, \theta[V_{n-1}, V_n]\}$ with $\theta \in \{\cup_p, \cap_p, -_p\}$ and $\forall i = 1, \ldots, n, V_i \in \{M_{Em}\}$.

From each computation path associated with a mediation entity, a set of possible computing expressions can be derived. The enumeration of the set of expressions, which can be derived from a path, consists in identifying all possible ordering combinations of operations

corresponding to the edges of the path C. For example, consider $C = \{\cup_p [V_1, V_2], \cap_p [V_2, V_3]\}$. Two different interpretations of this path leads to two different expressions $Exp_1 = (V_1 \cup_p V_2) \cap_p V_3$ and $Exp_2 = V_1 \cup_p (V_2 \cap_p V_3)$.

The example given in Figure 5.6 shows the computation paths associated with the operation graph of Figure 5.5 and one of the possible computing expressions. The computation paths and the computing expressions derived from them are described in the following.

- $C_1 = \{\cup_p [V_{Movie1}, V_{Movie2}], \cup_p [V_{Movie2}, V_{Movie3}]\}$ originates

  $Exp_{11} = (V_{Movie1} \cup_p V_{Movie2}) \cup_p V_{Movie3}$

  $Exp_{12} = V_{Movie1} \cup_p (V_{Movie2} \cup_p V_{Movie3})$

- $C_2 = \{-_p [V_{Movie2}, V_{Movie3}], \cup_p [V_{Movie3}, V_{Movie1}]\}$ originates

  $Exp_{21} = (V_{Movie2} -_p V_{Movie3}) \cup_p V_{Movie1}$

- $C_3 = \{\cup_p [V_{Movie1}, V_{Movie3}], \cup_p [V_{Movie2}, V_{Movie3}]\}$ originates

  $Exp_{31} = (V_{Movie1} \cup_p V_{Movie3}) \cup_p V_{Movie2}$

  $Exp_{32} = V_{Movie1} \cup_p (V_{Movie2} \cup_p V_{Movie2})$

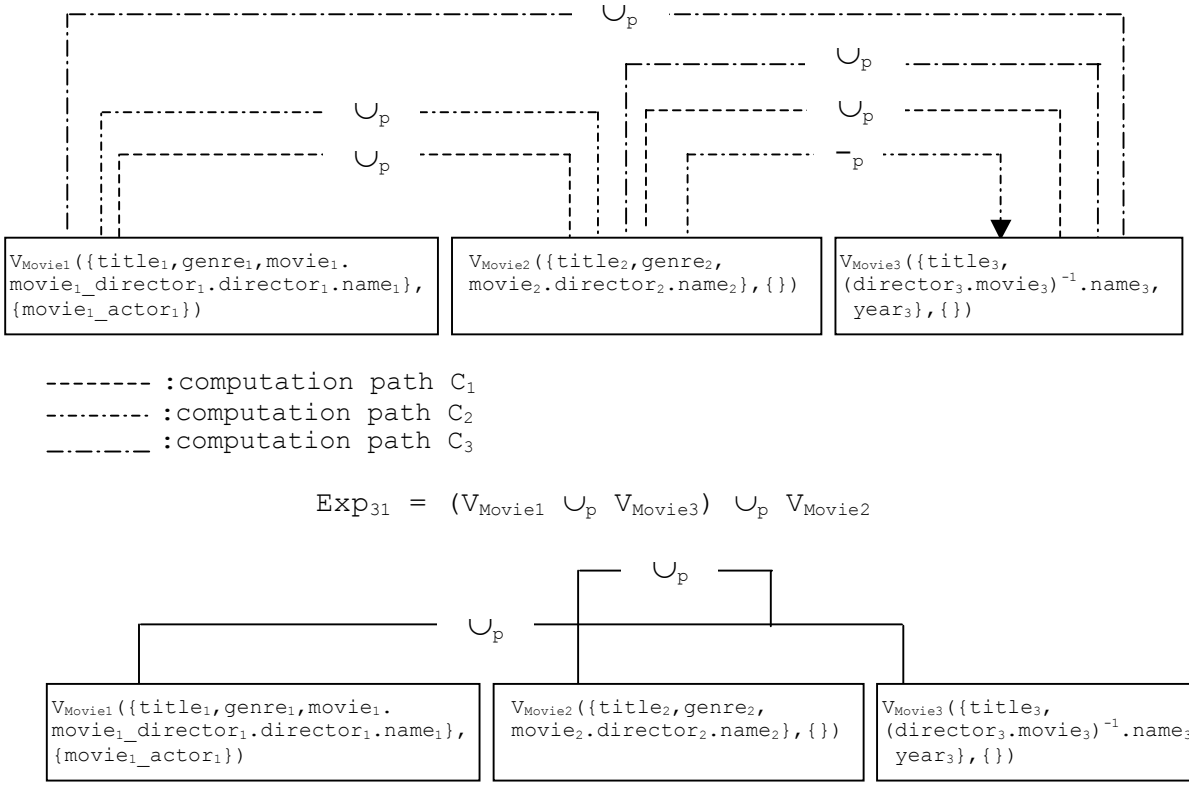

Figure 5.6 – Example of computation paths and computing expressions

A computation path includes only the set of operations which are necessary to compute a mediation entity. The ordering which these operations will be executed is determined by the computing expressions.

The enumeration of the computing expressions associated to a computation path, although it is always possible, can be a very complex task. Moreover, for each mediation entity, the set of possible expressions may have different semantics. As proposed in [Kedad et al. 1999] we can improve this approach by using some complementary knowledge such as heuristics, source quality or more specific user requirements. The use of this knowledge may considerably reduce the solution space. Example of source quality may be the preference given to one source among several equivalent because of the highest confidence, accuracy or completeness put on this source. Example of user requirement that can be considered may be an explicite indication to take systematically all sources; in which case the set of solutions elaborated with several equivalent sources is reduced by eliminated solution with single sources or subset of sources. A more detailed explanation about the derivation of computing expressions from computation paths as well as their definition is out of the scope of this work.

5.5 Computing mediation entities from mediation queries

As presented in Figure 5.7, each mediation entity is a view of data distributed in different data sources and is derived from the mapping views associated with E_m. The mapping views $\{V_{E11}, \ldots, V_{E1n}\}$ associated with a mediation entity E_m are presented in an operation graph, which describes the possible operators that can be applied bewteen them. Each mapping view V_{Eij} has a mapping query $q(V_{Eij})$ to compute its contents, which is derived from the contents of the source entity E_{ij}, such that V_{Eij} is derived from E_{ij}. The contents of a source entity E_{ij} is an XML view ($q(E_{ij})$) of the data actually stored in a given data source S_j.

Both mediation queries and mapping queries return collections of XML elements. The result of a mediation query $Q(E_m)$ is a collection of XML elements obtained through the integration of XML elements stored in the data sources. The result of a given mapping query $q(V_{Eij})$ is a subset of the collection of XML elements that are instances of the source entity E_{ij}, such that V_{Eij} is a mapping view derived from E_{ij}.

As defined in the X-Entity model, a mediation entity consists of a set of attributes and it is associated to a set of containment relationships. To compute a mediation entity E_m we will first compute the integrated instances that compose the mediation entity and, subsequently, for each containment relationship $R_m(E_m, E_m')$ we compute the integrated instances of the entity E_m'. In the remaining of this section, we present an example to illustrate the process of computing mediation entities from mediation queries.

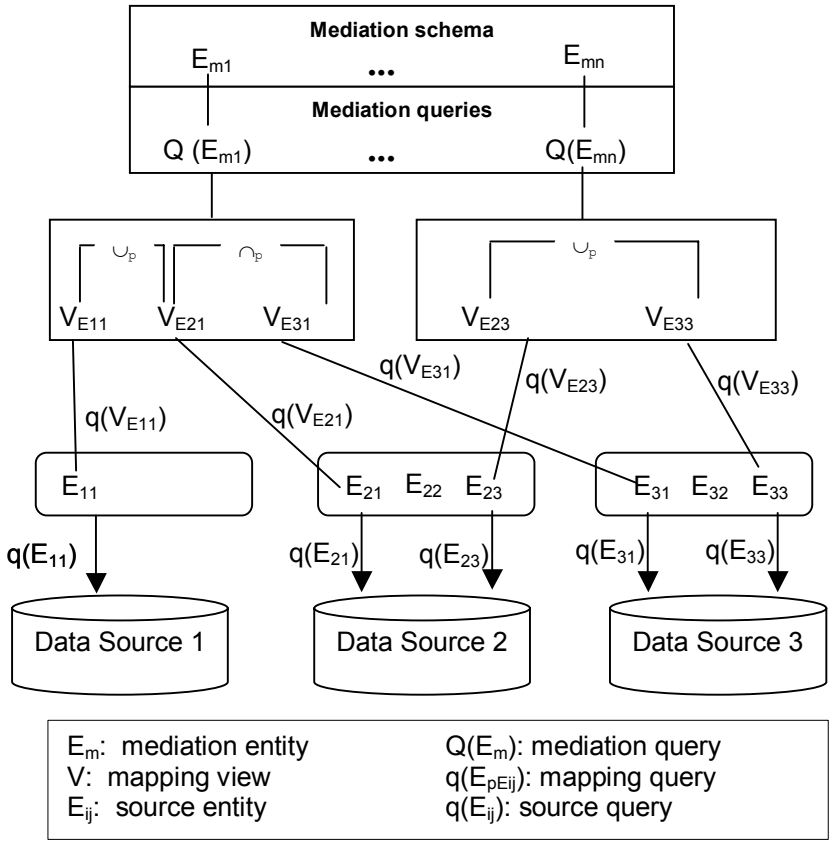

Figure 5.7 – Description of mediation queries and mapping queries

Example: Consider the source schemas and the mediation schema presented below.

```
Mediation Schema Smed =
({movie_m({title_m, director_m},{movie_m_theater_m}),
  theater_m({name_m,phone_m, time_m},{})},
 {movie_m_theater_m(movie_m, theater_m, (1,N))})
```

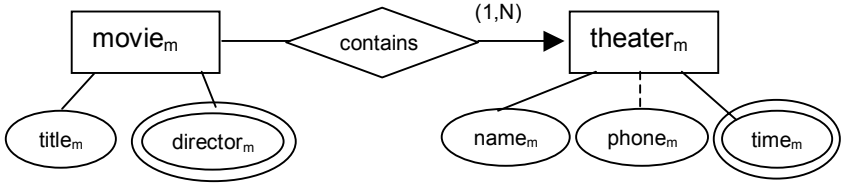

Figure 5.8 – Mediation schema S_{med}

```
Schema of data source S1 =
({movie1({title1, director1, actor1},{movie1_theater1}),
  theater1({name1, time1},{})},
 {movie1_theater1(movie1, theater1, (1,N))})
```

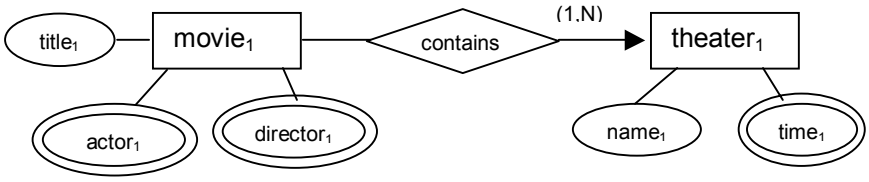

Figure 5.9 – Schema of data source S_1

```
Schema of data source S2 =
({movie2({title2, genre2},{movie2_theater2}),
  theater2({name2, phone2, time2},{})},
 {movie2_theater2(movie2, theater2, (1,N))})
```

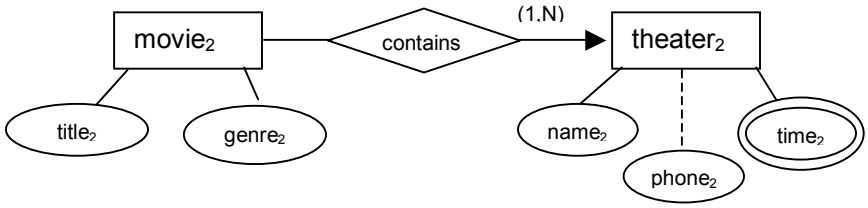

Figure 5.10 – Schema of data source S_2

Table 5.4 presents the correspondence assertions between the data sources S_1 and S_2.

Table 5.4 – Correspondence assertions between the local data sources schemas S_1 and S_2

Correspondence assertions between S_1 and S_2
$movie_1 \cong movie_2$
$movie_1 \cap movie_2 \neq \emptyset$
$movie_1.(title) \cong movie_2.(title)$
$theater_1 \cong theater_2$
$theater_1 \cap theater_2 \neq \emptyset$
$theater_1.(name_1) \cong theater_2.(name_2)$
$theater_1.time_1 \cong theater_2.time_2$

Figure 5.11 presents the operation graph which describes the mediation query of the mediation entity $movie_m$ ($Q(movie_m)$). This query specifies that the instances of the entity $movie_m$ correspond to the union of $movie_1$ instances stored in the data source S_1 with $movie_2$ instances stored in the data source S_2.

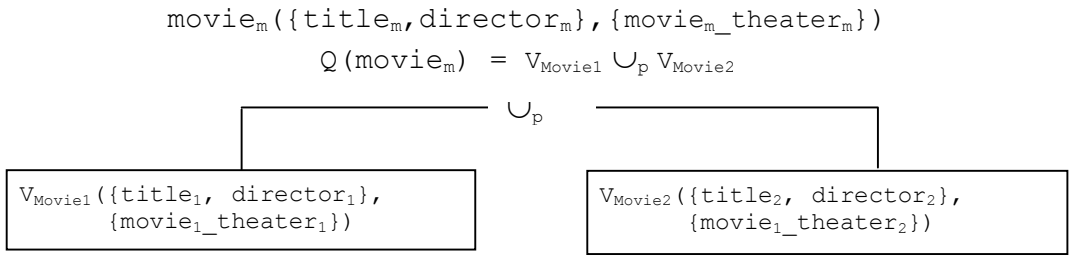

Figure 5.11 – Operation graph G_{moviem}

The first step to compute the mediation entity $movie_m$ consists in computing the mapping views V_{Movie1} and V_{Movie2}, i.e., the subqueries $q(V_{Movie1})$ and $q(V_{Movie2})$ must be sent to the data sources S_1 and S_2 respectively, such that V_{Movie1} and V_{Movie2} are mapping views derived from the source entities $movie_1$ and $movie_2$. As a result, the collections of XML elements presented in Figure 5.12 and Figure 5.13 are obtained. $C(movie_1)$ is the collection of $movie_1$

elements that are instances of the mapping view V_{Movie1} and $C(movie_2)$ is the collection of $movie_2$ elements that are instances of the mapping view V_{Movie2}.

The next step for computing the $movie_m$ entity consists of executing the mediation query $Q(movie_m)$ over the collections obtained in the earlier step ($C(movie_1)$ and $C(movie_2)$). After this, it is obtained the collection $C(movie_m)$ (Figure 5.14) composed by $movie_m$ instances, which represent the integration of $movie_1$ instances and $movie_2$ instances of the mapping views V_{Movie1} and V_{Movie2}. The element e_{m1}, for example, was obtained integrating the elements e_{11} and e_{21}. As defined earlier, when the union mapping operator is applied over two corresponding instances only one instance is obtained as a result, which represents the integration of the source instances. In our example, the element e_{m1} has the following subelements: $title_m$, derived from the $title_1$ and $title_2$ subelements of e_{11} and e_{21}, $director_m$, derived from the $director_2$ subelement of e_{11} and, $theater_1$ and $theater_2$ subelements, which belong to e_{11} and e_{21}, respectively.

$C(movie_1)$:

```
e11 = <movie1>
        <title1> Deus é brasileiro </title1>
        <director1> Cacá Diegues </director1>
        <theater1>
          <name1> Aldeota II </name1>
          <time1> 11h </time1>
          <time1> 13h40 </time1>
          <time1> 16h30 </time1>
          <time1> 19h </time1>
        </theater1>
        <theater1>
          <name1> Art Iguatemi I </name1>
          <time1> 15h40 </time1>
          <time1> 18h </time1>
          <time1> 20h20 </time1>
        </theater1>
      </movie1>
 e12 = <movie1>
        <title1> Gangs of New York </title1>
        <director1> Martin Scorsese </director1>
        <theater1>
          <name1> Art Iguatemi </name1>
          <time1> 14h20 </time1>
          <time1> 17h30 </time1>
          <time1> 20h40 </time1>
        </theater1>
        <theater1>
          <name1> North Shopping </name1>
          <time1> 13h50 </time1>
          <time1> 17h10 </time1>
          <time1> 20h30 </time1>
        </theater1>
      </movie1>
```

Figure 5.12 - Collection of $movie_1$ elements

C(movie$_2$):

```
e21 = <movie2>
        <title2> Deus é brasileiro </title2>
        <theater2>
          <name2> North Shopping I </name2>
          <time2> 13h50 </time2>
          <time2> 16h10 </time2>
          <time2> 18h30 </time2>
          <time2> 20h50 </time2>
          <phone2> 2875489 <phone2>
        </theater2>
        <theater2>
          <name2> Art Iguatemi I </name2>
          <time2> 15h40 </time2>
          <time2> 18h </time2>
          <time2> 20h20 </time2>
        </theater2>
      </movie2>
```

Figure 5.13 - Collection of $movie_2$ elements

As the entity $movie_m$ has a subentity ($theater_m$) we have to proceed with the computation of its integrated instances. We have to compute the $theater_m$ instances for each one of the $movie_m$ instances created in the earlier step. As showed in Figure 5.14 each $movie_m$ element is associated with a collection of $theater_1$ and $theater_2$ instances. Therefore, we have to execute the mediation query $Q(theater_m)$ over the collections $C(theater_1(e_{mi}))$ and $C(theater_2(e_{mi}))$, for each one of the $movie_m$ elements e_{mi}.

Figure 5.15 presents the operation graph that describes the mediation query $Q(theater_m)$. Observe that, in this case, it was not necessary to send subqueries to data sources S_1 and S_2, because the data was directly extracted from the integrated element e_{m1} (Figure 5.14). In some cases, if there are mapping views that were not "populated", then it would be necessary to send subqueries to the data sources. Executing the mediation query $Q(theater_m)$ over the collections $C(theater_1(e_{m1}))$ and $C(theater_2(e_{m1}))$, for example, the collection $C(theater_m(e_{m1}))$ is obtained, i.e., the collection of $theater_m$ elements associated with the element e_{m1}. The $theater_m$ elements are obtained integrating $theater_1$ and $theater_2$ elements stored in data sources S_1 and S_2. Figure 5.16 and Figure 5.17 show the collection of $theater_1$ and $theater_2$ elements, respectively. Figure 5.18 shows the collection $C(theater_m(e_{m1}))$.

In the final step, the elements in $C(theater_m(e_{m1}))$ replace the $theater_1$ and $theater_2$ elements of the element e_{m1} (Figure 5.19(a)) originating the element $movie_m$ presented in Figure 5.19 (b). The same process must be executed for all $movie_m$ instances in $C(movie_m)$.

C(movie$_m$):

```
e_m1 = <movie_m>
        <title_m> Deus é brasileiro </title_m>
        <director_m> Cacá Diegues </director_m>
        <?-- theater elements originated from data source S1 -->
        <theater_1>
          <name_1> Aldeota II </name_1>
          <time_1> 11h </time_1>
          <time_1> 13h40 </time_1>
          <time_1> 16h30 </time_1>
          <time_1> 19h </time_1>
        </theater_1>
        <theater_1>
          <name_1> Art Iguatemi I </name_1>
          <time_1> 15h40 </time_1>
          <time_1> 18h </time_1>
          <time_1> 20h20 </time_1>
        </theater_1>
        <?-- theater elements originated from data source S2 -->
        <theater_2>
          <name_2> Art Iguatemi I </name_2>
          <time_2> 15h40 </time_2>
          <time_2> 18h </time_2>
          <time_2> 20h20 </time_2>
          <phone_2> 2614890 </phone_2>
        </theater_2>
        <theater_2>
          <name_2> North Shopping I </name_2>
          <time_2> 13h50 </time_2>
          <time_2> 16h10 </time_2>
          <time_2> 18h30 </time_2>
          <time_2> 20h50 </time_2>
          <phone_2> 2875489 <phone_2>
        </theater_2>
      </movie_M>
e_m2 = <movie_m>
        <title_m> Gangs of New York </title_m>
        <director_m> Martin Scorsese </director_m>
        <?-- theater elements originated from data source S1 -->
        <theater_1>
          <name_1> Art Iguatemi </name_1>
          <time_1> 14h20 </time_1>
          <time_1> 17h30 </time_1>
          <time_1> 20h40 </time_1>
        </theater_1>
        <theater_1>
          <name_1> North Shopping </name_1>
          <time_1> 13h50 </time_1>
          <time_1> 17h10 </time_1>
          <time_1> 20h30 </time_1>
        </theater_1>

</movie_m>
```

Figure 5.14 - Collection of movie$_m$ elements

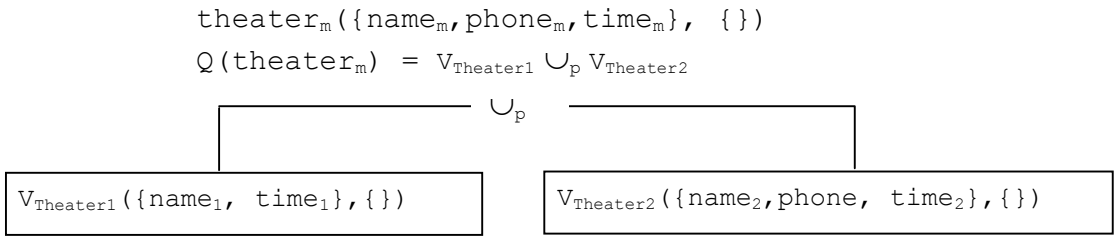

Figure 5.15 – Operation graph $G_{theaterm}$

C(theater$_1$(e$_{m1}$)):

```
<theater1>
  <name1> Aldeota II </name1>
  <time1> 11h </time1>
  <time1> 13h40 </time1>
  <time1> 16h30 </time1>
  <time1> 19h </time1>
</theater1>
<theater1>
  <name1> Art Iguatemi I </name1>
  <time1> 15h40 </time1>
  <time1> 18h </time1>
  <time1> 20h20 </time1>
</theater1>
```

Figure 5.16 - Collection of XML elements theater$_1$

C(theater$_2$(e$_{m1}$)):

```
<theater2>
  <name2> Art Iguatemi I </name2>
  <time2> 15h40 </time2>
  <time2> 18h </time2>
  <time2> 20h20 </time2>
  <phone2> 2614890 <phone2>
</theater2>
<theater2>
  <name2> North Shopping I </name2>
  <time2> 13h50 </time2>
  <time2> 16h10 </time2>
  <time2> 18h30 </time2>
  <time2> 20h50 </time2>
  <phone2> 2875489 <phone2>
</theater2>
```

Figure 5.17 - Collection of XML elements theater$_2$

C(theater$_m$(e$_{m1}$)):

```
<theaterm>
  <namem> Aldeota II </namem>
  <timem> 11h </timem>
  <timem> 13h40 </timem>
  <timem> 16h30 </timem>
  <timem> 19h </timem>
</theaterm>
<theaterm>
  <namem> Art Iguatemi I </namem>
  <timem> 15h40 </timem>
  <timem> 18h </timem>
  <timem> 20h20 </timem>
  <phonem> 2614890 <phonem>
</theaterm>
<theaterm>
  <namem> North Shopping I </namem>
  <timem> 13h50 </timem>
  <timem> 16h10 </timem>
  <timem> 18h30 </timem>
  <timem> 20h50 </timem>
  <phonem> 2875489 <phonem>
</theaterm>
```

Figure 5.18 - Collection of XML elements theater$_m$(e$_{m1}$)

```
<movie_m>
  <title_m> Deus é brasileiro </title_m>
  <director_m> Cacá Diegues </director_m>
  <?-- theater elements originated
   from data source S1 -->
  <theater_1>
    <name_1> Aldeota II </name_1>
    <time_1> 11h </time_1>
    <time_1> 13h40 </time_1>
    <time_1> 16h30 </time_1>
    <time_1> 19h </time_1>
  </theater_1>
  <theater_1>
    <name_1> Art Iguatemi I </name_1>
    <time_1> 15h40 </time_1>
    <time_1> 18h </time_1>
    <time_1> 20h20 </time_1>
  </theater_1>
  <?-- theater elements originated
   from data source S2 -->
  <theater_2>
    <name_2> Art Iguatemi I </name_2>
    <time_2> 15h40 </time_2>
    <time_2> 18h </time_2>
    <time_2> 20h20 </time_2>
    <phone_2> 2614890 <phone_2>
  </theater_2>
  <theater_2>
    <name_2> North Shopping I </name_2>
    <time_2> 13h50 </time_2>
    <time_2> 16h10 </time_2>
    <time_2> 18h30 </time_2>
    <time_2> 20h50 </time_2>
    <phone_2> 2875489 <phone_2>
  </theater_2>
</movie_m>
```

Figure 5.19(a) - Element e_{m1} before the integration of the $theater_1$ and $theater_2$ elements

```
<movie_m>
  <title_m> Deus é brasileiro </title_m>
  <director_m> Cacá Diegues </director_m>
  <theater_m>
  <name_m> Aldeota II </name_m>
  <time_m> 11h </time_m>
  <time_m> 13h40 </time_m>
  <time_m> 16h30 </time_m>
  <time_m> 19h </time_m>
</theater_m>
<theater_m>
  <name_m> Art Iguatemi I </name_m>
  <time_m> 15h40 </time_m>
  <time_m> 18h </time_m>
  <time_m> 20h20 </time_m>
  <phone_m> 2614890 <phone_m>
</theater_m>
<theater_m>
  <name_m> North Shopping I </name_m>
  <time_m> 13h50 </time_m>
  <time_m> 16h10 </time_m>
  <time_m> 18h30 </time_m>
  <time_m> 20h50 </time_m>
  <phone_m> 2875489 <phone_m>
</theater_m>
</movie_m>
```

Figure 5.19 (b) - Element e_{m1} after the integration of the $theater_1$ and $theater_2$ elements

The following algorithm summarizes the process of computing a mediation entity E_m over a set of data sources $S = \{S_1, \ldots, S_n\}$. In the following consider:

$Q(E_m)$: mediation query of E_m
$G_{Em}(M_{Em}, O_{Em})$: operation graph of E_m
$M_{Em} = V_{E1}, \ldots, V_{En}$: set of mapping views associated with E_m
$q(E_{pEi})$: subquery to compute E_{pEi}

```
compute_mediation_entity(Em, S)

For each VEi ∈ MEm
   C(EpEi) := execute q(VEi)
C(Em) := execute Q(Em) over {C(EpE1),...,C(EpEn)}
If Em.R ≠ ∅ Then
For each em ∈ C(Em)
   For each Rm(Em,Em') ∈ Em.R
      C(Em'(em)) := compute_child_entity(em, Ed')
      update(em, C(Em'(em)))
return C(Em)
```

Initially, the algorithm executes each one of the mapping queries ($q(V_{Ei})$) associated with mapping views of E_m in order to retrieve the source instances. Next, the mediation query is executed over the source instances. If the mediation entity has one or more containment relationships then an auxiliar function, called `compute_child_entity`, is called to compute the integrated instances for each one of the subentities. It is important to note that this process must be executed for each integrated instance of the mediation entity. When the integrated instances of a subentity are computed the integrated instance of the mediation entity has to be updated in order to replace the source instances by the integrated instances ($update(e_m, C(E_m'(e_m)))$).

5.6 Computing user queries from mediation entities

In this section we describe how a user query can be computed using the mediation entities available in the mediation schema. For a given user query against the mediation schema, the mediator tries to find combinations of mediation entities that allows the computation of the user query. We call such combination a query plan. Based on the mediation queries, previously defined, and on the query plan the system computes at run-time the most appropriate rewriting for answering a user query. As discussed in Chapter 3, the mediator module responsible for computing user queries is the query manager.

It is important to observe that when a mediation query is executed it returns all the elements that are instances of the corresponding mediation entity. However, a user query may specify some conditions that constraint the expected answer. Thus, only the elements which satisfy the constraint must be returned in the answer. In the following sections, we will illustrate this by an example.

We use as an example the mediation schema and the source schemas introduced in section 5.3 and reviewed as follows. Figures 5.20 and 5.21 show the mediation queries and operation graphs associated with the mediation entities. We also present three XML documents that are XML views of data stored in the data source S_1, S_2 and S_3 (Figures 5.22, 5.23 and 5.24). We show the source data using XML views, instead of their original format, in order to facilitate the understanding of the user queries execution process.

```
Mediation Schema S_med =
({movie_m({title_m, genre_m, year_m, director_m}, {movie_m_actor_m}),
  actor_m({name_m, nationality_m},{})},
 {movie_m_actor_m(movie_m,actor_m, (1,N))})
```

Schema of data source S_1 =
({$movie_1$({$title_1$,$duration_1$, $genre_1$},{$movie_1$_$actor_1$, $movie_1$_$director_1$}),
  $actor_1$({$name_1$, $nationality_1$},{}),
  $director_1$({$name_1$, $nationality_1$},{})},
 {$movie_1$_$actor_1$($movie_1$,$actor_1$,(1,N)),
  $movie_1$_$director_1$($movie_1$,$director_1$, (1,N))})

Schema of data source S_2 =
({$movie_2$({$title_2$,$genre_2$,$actor_2$},{$movie_2$_$director_2$}),
  $director_2$({$name_2$, $nationality_2$}, {})},
 {$movie_2$_$director_2$($movie_2$, $director_2$,(1,N))})

Schema of data source S_3 =
({$director_3$({$name_3$,$nationality_3$},{$director_3$_$movie_3$}),
  $movie_3$({$title_3$,$year_3$},{ })},
 {$director_3$_$movie_3$($director_3$,$movie_3$,(1,N))})

$movie_m$({$title_m$, $genre_m$, $year_m$, $director_m$}, {$movie_m$_$actor_m$})

$Q(movie_m) = (V_{Movie1} \cup_p V_{Movie3}) \cup_p V_{Movie2}$

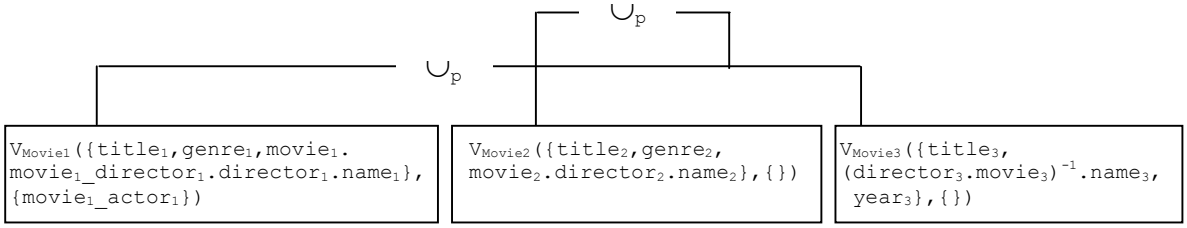

Figure 5.20 – Mediation query Q($movie_m$) and operation graph G_{moviem}

$actor_m$({$name_m$, $nationality_m$},{})

$Q(actor_m) = V_{Actor1}$

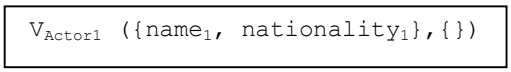

Figure 5.21 – Mediation query Q($actor_m$) and operation graph G_{actorm}

As finding query plans to answer user queries is out of the scope of this work, in the following examples, we suppose that a query plan was already generated.

User query 1: Suppose a user query asking for all movies whose director is "Steven Spielberg".

When a mediation query is executed it returns all the elements that are instances of the corresponding mediation entity. However, the user query may specify some conditions that constraint the expected answer. Considering our example about movies, the user query just requests the movies directed by "Steven Spielberg". Thus, only the movie elements where the subelement director has value "Steven Spielberg" must be returned in the answer. To solve this problem, we add a condition in the mapping queries in order to retrieve only the elements that satisfy the constraint specified in the user query.

q(V_{Movie1}(director$_1$.name$_1$ = "Steven Spielberg"))

$q(V_{Movie1}(movie_2.director_2.name_2 = \text{“Steven Spielberg”}))$

$q(V_{Movie3}((director_3.movie_3)^{-1}.name_3 = \text{“Steven Spielberg”}))$

After the execution of the mapping queries presented above and the integration of their results, we obtain the following collection of $movie_m$ elements:

```
<moviem>
  <titlem> Minority Report </titlem>
  <genrem> adventure </genrem>
  <directorm> Steven Spielberg </directorm>
  <yearm>2002 </yearm>
</moviem>
<moviem>
  <titlem> Men in Black II </titlem>
  <directorm> Steven Spielberg </directorm>
</moviem>
<moviem>
  <titlem> Jaws  </titlem>
  <directorm> Steven Spielberg </directorm>
  <yearm> 1975 </yearm>
</moviem>
```

```
<movies_list1>
 <movie1>
  <title1> Spider-man </title1>
  <genre1> adventure </genre1>
  <duration1> 121 minutes </duration1>
  <director1>
    <name1> Sam Raimi </name1>
  </director1>
  <actor1>
    <name1> Tobey Maguire </name1>
  </actor1>
  <actor1>
    <name1> Kirsten Dunst</name1>
  </actor1>
 </movie1>
 <movie1>
  <title1> Deus é Brasileiro </title1>
  <director1>
    <name1> Cacá Diegues </name1>
    <nationality1> brasileiro
    </nationality1>
  </director3>
  <actor1>
    <name1> Antonio Fagundes </name1>
  </actor1>
 </movie1>
</movies_list1>
```

Figure 5.22 – XML view of the data source S_1

```
<movies_list2>
 <movie2>
  <title2> Star Wars: Episode II -
           Attack of the Clones
  </title2>
  <genre2> science fiction</genre2>
  <director2>
    <name1> George Lucas </name2>
    <nationality2> american
    </nationality2>
  </director2>
  <actor2> Ewan McGregor </actor2>
  <actor2> Natalie Portman </actor2>
 </movie2>
 <movie2>
  <title2> Spider-man </title2>
  <genre2> adventure </genre2>
  <director2>
    <name2> Sam Raimi </name2>
  </director2>
  <actor2> Tobey Maguire </actor2>
  <actor2> Kirsten Dunst </actor2>
 </movie2>
 <movie2>
  <title2> Minority Report </title2>
  <genre2> adventure </genre2>
  <actor2> Tom Cruise </actor2>
  <actor2> Tea Leoni  </actor2>
 </movie2>
 <movie2>
  <title2> Men in Black II </title2>
 </movie2>
 <movie2>
   <title2> Jaws  </title2>
 </movie2>
</movies_list2>
```

Figure 5.23 – XML view of the data source S_2

```
<directors_list3>
 <director3>
  <name3> Steven Spielberg  </name3>
  <nationality3>american  </nationality3>
  <movie3>
    <title3> Minority Report </title3>
    <year3>2002 </year3>
  </movie3>
  <movie3>
    <title3> Jaws  </title3>
    <year3> 1975 </year3>
  </movie3>
 </director3>
 <director3>
  <name3> George Lucas </name3>
   <movie3>
    <title3> Star Wars: Episode II – Attack of the Clones </title3>
  </movie3>
 </director3>
</directors_list3>
```

Figure 5.24 – XML view of the data source S_3

User query 2: Suppose a user query asking for all title movies together with their related directors. The relevant mediation entity to answer this query is $movie_m$. Therefore, to answer this query we use the mediation query $Q(movie_m)$, which specifies that the $movie_m$ instances are obtained through the union of $movie_1$, $movie_2$ and $movie_3$ instances obtained through the mapping queries $q(E_{pMovie1})$, $q(E_{pMovie2})$ and $q(E_{pMovie3})$, respectively. Executing these subqueries and integrating their results we obtain the following $movie_m$ elements:

```
<moviem>
  <titlem> Star Wars: Episode II – Attack of the Clones </titlem>
  <directorm> George Lucas </directorm>
</moviem>
<moviem>
  <titlem> Spider-man </title1>
  <directorm> Sam Raimi </directorm>
</moviem>
<moviem>
  <titlem> Minority Report </titlem>
  <directorm> Steven Spielberg </directorm>
</moviem>
<moviem>
  <titlem> Men in Black II </titlem>
  <directorm> Steven Spielberg </directorm>
</moviem>
<moviem>
  <titlem> Jaws  </titlem>
  <directorm> Steven Spielberg </directorm>
</moviem>
<moviem>
  <titlem> Deus é Brasileiro </titlem>
  <directorm> Cacá Diegues </directorm>
</moviem>
<moviem>
  <titlem> Star Wars: Episode I – The  Phantom Menace </titlem>
  <directorm> George Lucas </directorm>
</moviem>
<moviem>
  <titlem> Star Wars: Episode II – Attack of the Clones </titlem>
  <directorm> George Lucas </directorm>
</moviem>
```

5.7 Concluding remarks

In this chapter we presented the process for generating XML-based mediation queries. We extended the approach proposed by Kedad & Bouzeghoub [Kedad et al. 1999], which specifies how to generate computing expressions for relational views, by: i) redefining the process of identifying the relevant source entities, ii) specifying new operators to be applied between the mapping views and iii) redefining the process of generating computing expressions.

In our approach, the mediation schema is represented by an X-Entity schema, which is composed by entities and relationships among them. Therefore, the process of mediation queries generation consists in discovering a computing expression for each entity in the mediation schema. The use of mediation queries improves the efficiency of query planning because the different possible ways of obtaining data for a mediation entity are previously defined.

The first task of the process of mediation queries discovering consists in identifying the source entities that are relevant to the computation of a given mediation entity. Informally, a source entity E_i is relevant to compute a mediation entity E_m if E_i and E_m are semantically equivalent, i.e., they represent the same concept in the real world. As an entity is composed by attributes and relationship types then it is necessary to identify the attributes and the relationship types of the source entities which are relevant to compute the mediation entity. The links between the mediation entity and its relevant source entities are represented through mapping views. Besides attributes and relationships, a mapping view may contain attribute derivation paths and subentities derivation paths. The next task of the process of mediation queries discovering is the identification of the possible operators to be applied between the mapping views.

The last task of the process of mediation queries discovering consists in determining the computation paths associated with a mediation entity and generating the computing expressions corresponding to each computation path. The generation of a mediation entity expression is the identification of the set of all possible orderings of the operations contained in the computation path. The generation of a mediation entity expression can be a very complex and time-consuming task, because there can be many different computation paths and for each computation path, distinct computing expressions may be obtained. Therefore, some heuristics must be proposed in order to facilitate this task.

We also discussed the process of computing mediation entities from mediation queries. During the computation of a mediation entity, the integration of corresponding instances is

performed, i.e., instances corresponding to the same object in the real world, but stored in different data sources. The identification of corresponding instances is done through the values of their common identifier (defined by the mapping attributes) and the integration of the instances is done by the mapping operators. Besides eliminating redundancies, these operators perform the fusion of semantically equivalent instances.

As the structure of XML data is more flexible than the structure of conventional data, then the process of computing mediation entities is more complex. In the relational model, for example, a relation is composed by a set of tuples having the same schema and consisting of attributes, which are atomics and monovalued. On the other hand, XML elements may be composed by other elements and attributes, and elements with the same type may have different structures. The process of computing mediation entities has two main tasks: the computation of the mediation entity instances through the integration of its corresponding mapping views instances and, next, for each subentity of the mediation entity, it is performed the computation of its own instances through the integration of the instances of its corresponding mapping views.

At the end of the mediation queries generation process, each mediation entity will be associated with a mediation query, which is represented by an operation graph composed by a set of mapping views and operations among them. As we will present in the next chapter, this high-level representation facilitates the identification of the mediation entities that are affected by a data source schema change and that, consequently, should be rewritten. A mediation entity will be affected by an entity source change if one of its mapping views will be affected by the data source change. Moreover, we will see that it is easier to propagate data source changes into mediation queries. Now, the problem of propagating the source changes or users' requirements changes into the mediation queries consists, first, in propagating these changes into the mapping views, and secondly, in modifying the mediation queries in order to take into account the modifications in the set of mapping views. In the next chapter, we present in details the process of managing the evolution of mediation queries.

Chapter 6

Managing the Evolution of Mediation Queries

6.1 Introduction

One of the main challenges in data integration systems is to maintain the mediation schema consistent with the users' requirements evolution and to maintain the mediation queries consistent both with the mediation schema evolution and with source evolution. The evolution of the mediation schema is in many aspects similar to the schema evolution problem in traditional databases. The novel and complex problem of evolution in mediation systems is the change of the mediation queries, especially in the GAV approach where mediation queries are very sensitive to changes in source descriptions. In [Lóscio et al. 2002b] we present a solution to the problem of mediation queries maintenance for data integration systems which adopt the relational model as the common data model.

The statement of this evolution problem is as follows: given a change event occurring at the source level or at the user level, how to propagate this change into the mediation queries. In this context, mainly two kinds of evolution have to be dealt within a mediation-based system:

i) *the evolution of the user needs* - it may consist in adding, removing or modifying an user requirement. These changes impact the mediation schema by adding, modifying or deleting an element from the mediation schema. If these changes can be reflected in the mediation queries, the modifications on the mediation schema are committed; otherwise the user is informed that his or her new requirements cannot be satisfied, and

ii) *the evolution of the data source schemas* - if a change occurs in a source schema, it has to be propagated to the mediation queries. The later are modified if the source elements on which they were defined are modified or when a source element is added or deleted.

In this chapter, we describe our approach to manage the evolution of mediation queries. We start by introducing the problem of propagating users' requirements and data source schemas changes to the mediation schema and the mediation queries. We also present the X-entity schema change operations and the propagation primitives used to update the mediation level. Next, we introduce the set of rules proposed for propagating the data source schemas changes and the users' requirements changes into the mapping views and the associated operations. We also present the propagation processes used to propagate schema changes to the mediation level. Finally, we summarize the chapter with some concluding remarks.

6.2 Propagation of schema changes to the mediation queries

We propose a solution to the problem of managing the evolution of mediation queries in dynamic environments. Changes to mediation queries may be due to changes in the users' requirements or data source schemas. Figure 6.1 describes the impact of these changes in the mediation level. In this context, change management policies are necessary in order to maintain the consistency between the data sources level, the mediation level and the user level.

Addition of new data sources and modifications in the data source schemas are changes that must be propagated both to the user level and the mediation level. In the first case, the propagation involves an analysis of existing users' requirements to identify relevant source entities to compute user entities (E_u), i.e., entities participating in the users' requirements schema. Operation graphs for such entities can be incrementally created according to the evolution of the data sources. Moreover, whenever a mediation query can be generated for a user entity then it can be inserted in the mediation schema.

The propagation of data source schemas changes to the mediation level includes also the propagation of data sources removal and consists mainly in changing the mediation queries when the changes raised at the source level still allow the computation of the mediation entities (E_m), provided some changes done on the mediation queries. However, some mediation entities may become no longer computable concerning to the changes raised at the source level. Therefore, in these cases, the mediation entities must be removed from the mediation schema. We consider that the mediation schema has only completely computable user entities, i.e., entities whose attributes and subentities can be computed from the data sources.

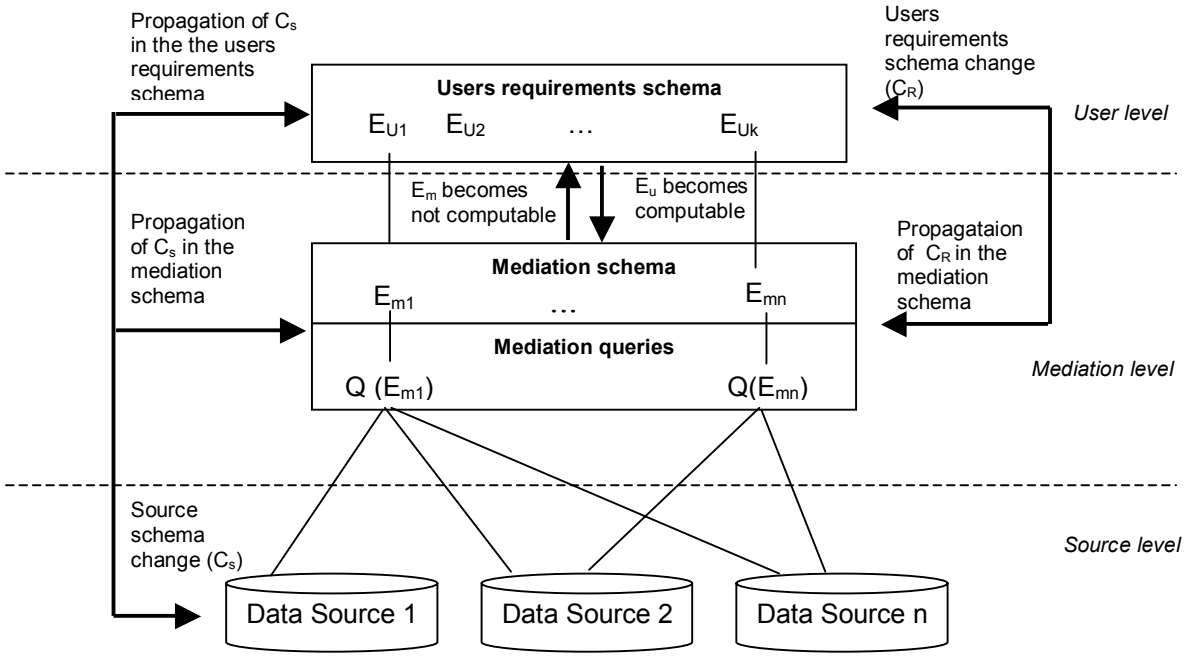

Figure 6.1 – Propagation of data source schemas changes and users' requirements changes

Concurrently with the evolution of the data source schemas, the users' requirements may continue to change. This may happen during system development as well as when the system is initialized. The evolution of the users' requirements schema originates directly change operations in the mediation schema. If these changes can be reflected in the mediation queries, the modifications on the mediation schema are committed; otherwise the user is informed that his or her new requirements cannot be satisfied. This occurs when the requirement can not be computable from the data available in the data sources. Each change raised in the mediation schema may lead to the redefinition of some mediation queries or the generation of new ones.

Since the mediation queries are generated using the algorithm presented in Chapter 5, each mediation entity is associated with a set of mapping views and each mediation query consists of a set of operations applied to the mapping views. Consequently, the problem of propagating users' requirements changes and data source schemas changes into the mediation queries consists, first in propagating these changes into the mapping views, and secondly in modifying the mediation queries in order to take into account the modifications in the set of mapping views.

In the remaining of this section, we will introduce the X-Entity schema change operations, the propagation primitives and the mapping view evolution rules.

6.2.1 X-Entity schema change operations

X-Entity schema change operations specify modifications that are performed in the local schemas or in the mediation schema and that must be propagated to the mediation queries. Local schema changes are detected by the *Conceptual Schema Manager* (cf. Chapter 3) through the comparison of two different versions of the same conceptual schema. On the other hand, changes in the mediation schema represent modifications in the users' requirements.

We propose the types of X-Entity change operations showed below. In the following, consider:

- `add_entity(E,S)`: this operation adds a new entity type `E` in the schema `S`. If the entity `E` has one or more containment relationships, then this operation may be followed by other operations `add_Entity(E' ,S)`, where `E'` is an entity type associated with `E` through a containment relationship `R`, and `add_relationship(R, S)` operations. If the entity `E'` does not exist in the schema `S`, then it is necessary to add it. Similarly, `R` is a new relationship and must be added into `S`. If the new entity type `E` is a subentity of another entity that already exists, then the operation `add_entity(E,S)` is followed by an operation `add_contains_rel(E", R')`, where `R'(E", E, (min, max))`.

  Example: Consider the schema S_1 presented in Figure 6.2(a). Suppose that we want to add an `actor` entity type, which has an `award` subentity, in the content of the `movie` entity. To do this, the following change operations must be executed over S_1:

  - `add_entity(actor({},{actor_award}),`S_1`)`
  - `add_entity(award({},{}),`S_1`)`
  - `add_relationship(actor_award(actor,award,(0,N)),`S_1`)`
  - `add_contains_rel(movie, movie_actor)`

  The schema presented in Figure 6.2 (b) is the new version of the schema S_1.

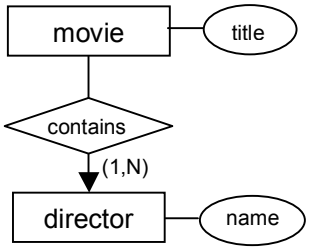

Figure 6.2(a) - Schema S_1

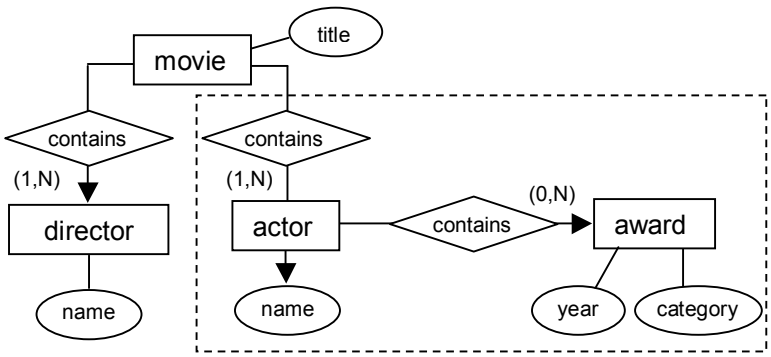

Figure 6.2(b) – New version of schema S_1

- remove_entity(E,S): this operation removes an existing entity E from the schema S. If the entity E has one or more containment relationships, then this operation may be followed by other operations remove_entity(E',S), where E' is a subentity of E. If there is not another entity X, such that "X contains E' " then E' may be removed from the schema S. Besides removing the entity E this operation also removes all containment relationships associated with it. If the removed entity type E is a subentity of another entity, E" then the operation remove_entity(E,S) is followed by an operation remove_contains_rel (E", R'), where R'(E", E, (min, max)).

  Example: Consider the schema S_2 presented in Figure 6.3(a). Suppose that we want to remove the award entity from the set of subentities[11] of the actor entity. To do this, the following change operations must be executed over S_2:

  – remove_entity(award,S_2)
  – remove_contains_rel(actor, actor_award)

  The schema presented in Figure 6.3(b) is a new version of the schema S_2.

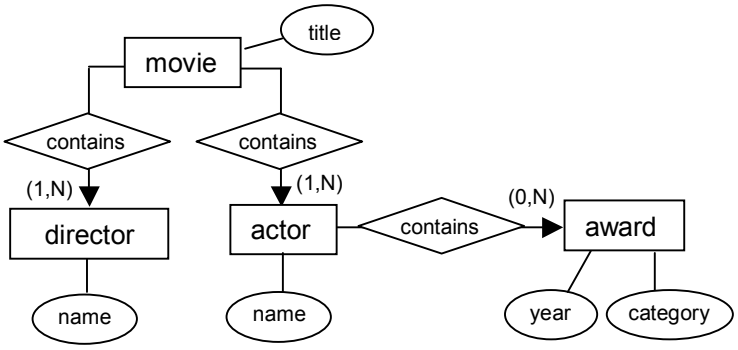

Figure 6.3(a) - Schema S_2

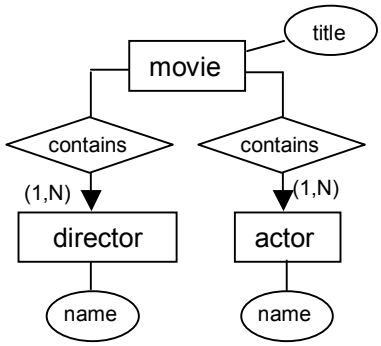

Figure 6.3(b) – New version of S_2

- add_attribute(E,A): this operation adds a new attribute A into the set of attributes of the entity E.

  Example: Consider the schema S_3 presented in Figure 6.4(a). Suppose that we want to add the genre and year attributes in the movie entity. To do this, the following change operations must be executed over S_3:

  – add_attribute(movie,year(string,(0,1)))
  – add_attribute(movie,genre(string,(0,1)))

  The schema presented in Figure 6.4(b) is the new version of the schema S_3.

[11] A subentity of an entity type E is an entity type E' which is associated with E through a containment relationship R(E,E',(min,.max)).

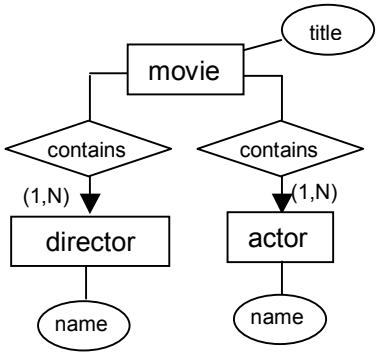

Figure 6.4(a) – Schema S_3

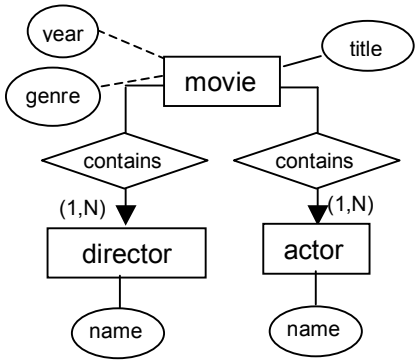

Figure 6.4(b) – New version of schema S_3

- `remove_attribute(E,A)`: this operation specifies the removal of an attribute `A` from the set of attributes of the entity `E`.

Example: Consider the schema S_4 presented in Figure 6.5(a). To remove the `genre` attribute from the `movie` entity type we perform the following change operation: `remove_attribute(movie,genre)`. The schema presented in Figure 6.5(b) is the new version of the schema S_4.

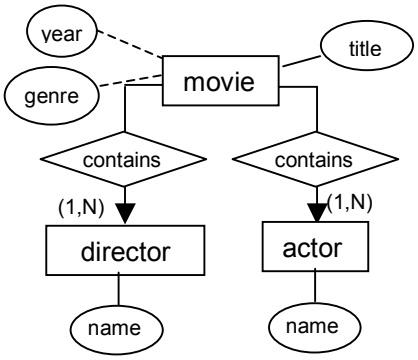

Figure 6.5(a) – Schema S_4

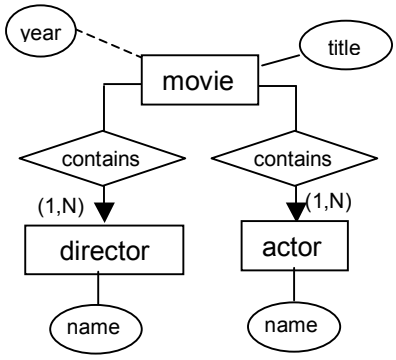

Figure 6.5(b) – New version of schema S_4

- `add_contains_rel(E,R)`: this operation adds a new containment relationship `R`, where `R(E,`E_k`,(min, max))`, into the set of containment relationships of the entity `E`. In this case, the entity E_k already exists and only a new containment relationship is inserted between `E` and E_k. As the relationship `R` is a new one, then an `add_relationship(R,S)` operation is executed to insert `R` into the schema `S`.

Example: Consider the schema S_5 presented in Figure 6.6(a). Suppose that we want to add a new containment relationship between `director` and `award`. To do this, we add a new containment relationship in the set of containment relationships of the `director` entity: `add_contains_rel(director,director_award)`. We also add the relationship `director_award` in the set of relationships of the schema `S`: `add_relationship`

director_award(director,award,(0,N)), S). The schema presented in Figure 6.6 (b) is a new version of the schema S_5.

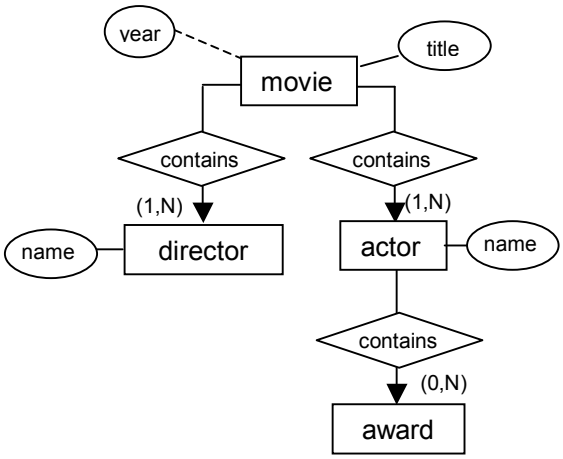

Figure 6.6(a) – Schema S_5

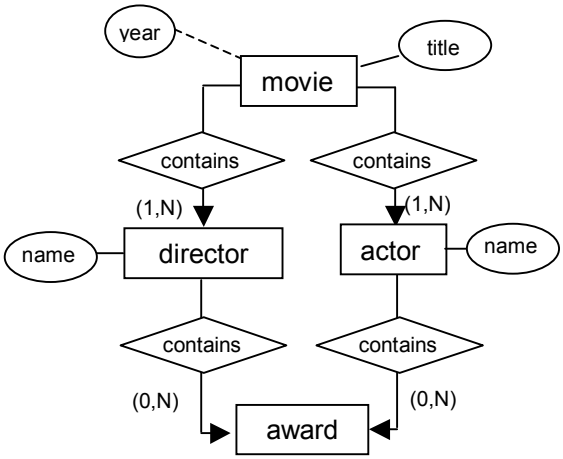

Figure 6.6(b) – New version of schema S_5

- remove_contains_rel(E,R): this operation removes the containment relationship R where $R(E,E_k,(min, max))$, from the set of containment relationship of the entity E. This operation is followed by an operation remove_relationship(R,S). Moreover, this operation may be followed by an operation remove_entity(E',S) if there is not another entity X, such that "X contains E' ".

  Example: Consider the schema S_6 presented in Figure 6.7(a). Suppose that we want to remove the containment relationship between actor and award. To do this, we execute the following change operation: remove_contains_rel(actor,actor_award). The schema presented in Figure 6.7(b) is the new version of the schema S_6.

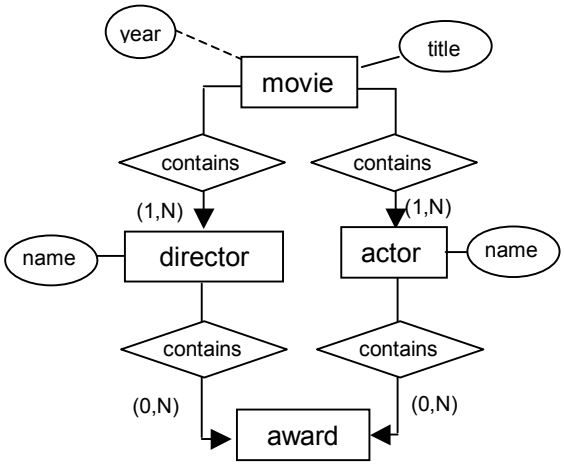

Figure 6.7(a) – Schema S_6

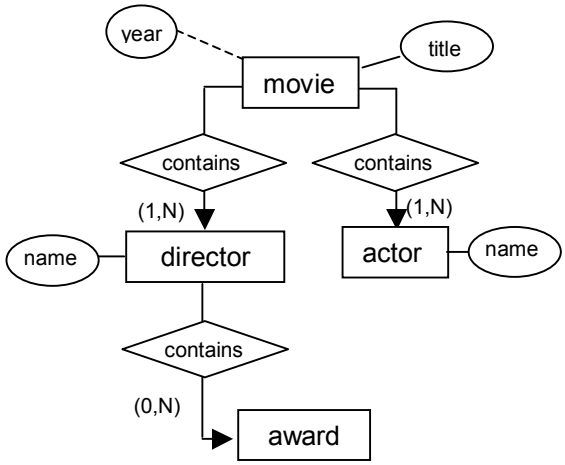

Figure 6.7(b) – New version of schema S_6

The removal or the addition of a data source can be represented in terms of a set of change operations, for example, the addition of a source can be viewed as a set of add_entity

operations and the removal of a data source can be viewed as a set of remove_entity operations. Table 6.1 summarizes the local schema change operations.

The set of X-Entity change operations may be extended with other operations, including modifications on the reference relationships and disjunction constraints.

Table 6.1 – X-Entity schema change operations

Change Operation	Definition
add_entity(E,S)	Adds a new entity type E into the schema S
remove_entity(E,S)	Removes an existing entity type E from the schema S
add_attribute(E,A)	Adds the attribute A into E.A
remove_attribute(E,A)	Removes the attribute A from E.A
add_contains_rel(E,R)	Adds the containment relationship R into E.R
remove_contains_rel(E,R)	Removes the containment relationship R from E.R
add_relationship(R,S)	Adds a new relationship type R into the schema S
remove_relationship(R,S)	Removes an existing relationship type R from the schema S

6.2.2 Propagation primitives

We consider that each mediation entity E_m is associated with an operation graph $G_{Em}(M_{Em}, O_{Em})$ corresponding to the mediation query defining E_m, where M_{Em} is the set of nodes of G_{Em}, representing the set of m-entities associated with E_m, and O_{Em} is the set of edges of G_{Em}, labeled with one of the mapping operators. If a change occurs in the data source schemas or in the mediation schema, some checking operations have to be performed on this graph to test if the computation path and therefore the mediation query associated with E_m are still valid. If not, new computation paths have to be determined and a new query has to be defined. Each entity E_m in the mediation schema is associated with two attributes, MAPSET_STATUS and OPSET_STATUS. These attributes have boolean values and represent the status of the set of mapping views associated with $E_m(M_{Em})$ and the set of candidate operations to combine these entities (O_{Em}). They determine if the set of mapping views associated with E_m (M_{Em}) and the set of candidates operations to combine these relations (O_{Em}) were modified during the propagation of the schema changes. These two attributes are set to False at the beginning of the propagation process, and they will be set to True if a change occurs in the set of mapping views or the set of operations respectively.

The propagation primitives, presented in Table 6.2, specify modifications and verifications which must be performed in the operation graphs and the corresponding mediation queries to reflect local schema change operations or mediation schema change operations. In the

following, G_{Em} denotes an operation graph corresponding to a mediation entity E_m. The last three primitives will be used in the algorithms describing the propagation processes and presented in section 6.5. The other primitives are used in the mapping view evolution rules.

Table 6.2 - Mediation queries propagation primitives

Mediation Level Propagation Primitive	Definition
search_operation(G_{Em})	Searches new operations for combining pairs of mapping views in the operation graph G_{Em}. If new operations are generated, then the attribute OPSET_STATUS is set to TRUE
remove_operations(G_{Em},V,A)	Removes all edges in the operation graph G_{Em} that become invalid because of the removal of the attribute A from the mapping view V. If at least one operation is removed, then the attribute OPSET_STATUS is set to TRUE
add_mapping(V, G_{Em})	Adds a mapping view V into the operation graph G_{Em} and assigns the TRUE value to the attribute MAPSET_STATUS. If G_{Em} does not exist then this primitive creates G_{Em} from V.
remove_mapping(V, G_{Em})	Removes the mapping view V from the operation graph G_{Em} and assigns the TRUE value to the attribute MAPSET_STATUS
search_computation_path(G_{Em})	Determines the computation paths associated with the operation graph G_{Em} (as described in Chapter 5)
generate_query(G_{Em}, Q)	Generates the set Q of computing expressions to compute the entity E_m using the operation graph G_{Em}
query_choice(Q, q)	Takes as input a set of possible queries Q and produces as output a single query q (the choice is made either by the designer or using some heuristics)

6.2.3 Mapping views evolution rules

Given a change represented by one of the schema change operations described in section 6.2, we will first propagate these changes in the set of mapping views associated with each entity of the mediation schema. To specify this propagation, we use event-condition-action (ECA) rules defined as follows:

- *Event (E)*: is a schema operation which represents the change (cf. Table 6.1),
- *Condition (C)*: is a condition related either to the local schemas or mediation schema metadata, or to the mapping views.
- *Action (A)*: is a sequence of propagation primitives which updates the mapping views and the corresponding operations to reflect the schema change (cf. Table 6.2).

In the next sections, we present the rules to propagate schema changes into mapping views. First, we introduce the rules to propagate data source schemas changes, then we describe the

rules to propagate users' requirements changes. The schema changes that have to be propagated are summarized below:

- add_entity(E,S)
- remove_entity(E,S)
- add_attribute(E,A)
- remove_attribute(E,A)
- add_contains_rel(E,R)
- remove_contains_rel(E,R)

In the following sections, consider:

- E_m: mediation entity
- V: mapping view
- E_i: source entity
- ADP: attribute derivation path
- EDP: entity derivation path
- G_{Em}: operation graph of the mediation entity E_m
- M_{Em}: is the set of mapping views corresponding to the mediation entity E_m
- $X(E_i)$: is the set of mapping attributes between the source entity E_i and the other source entities which originated mapping views belonging to M_{Em}
- $V = Exp(E_i)$: specifies that the mapping view V is derived from the source entity E_i

6.3 Using mapping views evolution rules to propagate data source schemas changes

We propose a set of rules to propagate data source schemas changes into mapping views. As described in Chapter 5, a mapping view specifies how to compute attributes and subentities of a mediation entity E_m from a source entity E_i. A mapping view $V(\{X_1,...,X_n\},\{Y_1,...,Y_m\})$ is a special entity type where X_i is an attribute or an attribute derivation path and Y_i is a relationship or an entity derivation path. So, the propagation of a data source schema change into a mapping view V may result in the addition or removal of an attribute, containment relationship or derivation path from V.

The mapping views evolution rules are classified according to the type of schema change operation. Each rule has a name and a parameter denoted E_m which represents a mediation entity. The instantiation of this parameter is detailed in the algorithms describing the propagation process, which will be presented in Section 6.5.

Example: we use as an example the mediation schema and the source schemas introduced in section 5.3 and reviewed in the following. Figures 6.8 and 6.9 show the mediation queries and operation graphs associated with the mediation entities. It is important to note that, in the most examples discussed in the following sections we will consider the original versions of the operation graphs (Figures 6.8 and 6.9) instead of the new versions obtained after the propagation of a source schema change.

Mediation Schema S_{med}:
  $movie_m$({$title_m$, $genre_m$, $year_m$, $director_m$}, {$movie_m\_actor_m$})
  $actor_m$({$name_m$, $nationality_m$},{})
  $movie_m\_actor_m$($movie_m$,$actor_m$,(1,N))

Schema of data source S_1:
  $movie_1$({$title_1$, $duration_1$, $genre_1$},{$movie_1\_actor_1$, $movie_1\_director_1$})
  $actor_1$({$name_1$, $nationality_1$},{})
  $director_1$({$name_1$, $nationality_1$},{})
  $movie_1\_actor_1$($movie_1$,$actor_1$,(1,N))
  $movie_1\_director_1$($movie_1$,$director_1$, (1,N))

Schema of data source S_2:
  $movie_2$({$title_2$,$genre_2$,$actor_2$},{$movie_2\_director_2$})
  $director_2$({$name_2$, $nationality_2$}, {})
  $movie_2\_director_2$($movie_2$, $director_2$,(1,N))

Schema of data source S_3:
  $director_3$({$name_3$,$nationality_3$},{$director_3\_movie_3$})
  $movie_3$({$title_3$,$year_3$},{})
  $director_3\_movie_3$($director_3$,$movie_3$,(1,N))

$movie_m$({$title_m$, $genre_m$, $year_m$, $director_m$}, {$movie_m\_actor_m$})

$Q(movie_m) = (E_{pMovie1} \cup_p E_{pMovie3}) \cup_p E_{pMovie2}$

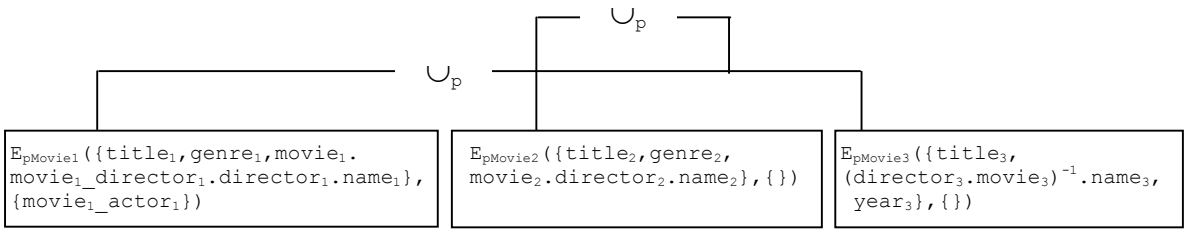

Figure 6.8 – Mediation query Q($movie_m$) and operation graph G_{moviem}

$actor_m$({$name_m$, $nationality_m$},{})

$Q(actor_m) = E_{pActor1}$

$E_{pActor1}$ ({$name_1$, $nationality_1$},{})

Figure 6.9 – Mediation query Q($actor_m$) and operation graph G_{actorm}

6.3.1 Adding an attribute into a source entity

The following ECA rules (Rule 1, Rule 2, Rule 3 and Rule 4) update the mapping views corresponding to the mediation entity E_m after an insertion of a new attribute A_k into a source entity E_i, as described in the following:

- **Rule 1:** checks if the new attribute A_k is semantically equivalent to one of the attributes of the mediation entity E_m and if there is a mapping view V associated with E_m over E_i. If the attribute A_k is semantically equivalent to one of the attributes of the entity E_m, this means that there is at least one mapping view V' containing the attribute A_k and derived from a source entity E_x which is distinct from E_i. If the condition is true, the attribute A_k must be inserted into the mapping view V. As a consequence of this, new operations must be searched. If the new attribute is a mapping attribute of the entity E_i, then after its addition it may be possible the application of a new mapping operator.

```
Rule 1(Em)
When add_attribute(Ei, Ak)
If ∃Amk ∈ Em.A | Em.Amk ≅ Ei.Ak
Then If ∃ V ∈ MEm | V = Exp(Ei)
     Then V.A := V.A ∪ {Ak}
          search_operation(GEm)
```

- **Rule 2:** checks if the new attribute A_k is semantically equivalent to one of the attributes of the mediation entity E_m and if there is no mapping view V derived from the source entity E_i. In this case, a new mapping view should be added into the operation graph G_{Em} and the operations between the new mapping view and the other mapping views in G_{Em} must be identified.

```
Rule 2(Em)
When add_attribute(Ei, Ak)
If ∃Amk ∈ Em.A | Em.Amk ≅ Ei.Ak
Then If V ∉ MEm | V = Exp(Ei)
     Then V.A := {Ak} ∪ X(Ei)
          add_mapping(V,GEm)
          search_operation(GEm)
```

- **Rule 3**: checks if there is a source entity E_j that is semantically equivalent to E_m and if there is a mapping view V associated with E_m over E_j. In the positive case, the condition part of the rule also verifies if there is an attribute derivation path (ADP) from the entity type E_j to the attribute A_k that is semantically equivalent to one of the attributes of the mediation entity E_m. If all conditions are true, the derivation path must be inserted into the mapping view V.

Rule 3(E_m)
When add_attribute(E_i, A_k)
If $\exists E_j \cong E_m \mid \exists V \in M_{Em} \wedge V = Exp(E_j)$
Then If $\exists A_{mk} \in E_m.A \mid E_m.A_{mk} \cong ADP$, where $ADP=E_j....E_i.A_k \vee ADP=(E_i....E_j)^{-1}.A_k$
Then $V.A := V.A \cup \{ADP\}$

- **Rule 4**: checks if there is a source entity E_j that is semantically equivalent to E_m and if there is no mapping view V associated with E_m over E_j. In the positive case, the condition part of the rule also verifies if there is an attribute derivation path from the entity type E_j to the attribute A_k that is semantically equivalent to one of the attributes of the mediation entity E_m. If all conditions are true, a new mapping view should be added in the operation graph G_{Em} and the operations between the new mapping view and the other mapping views in G_{Em} must be identified.

Rule 4(E_m)
When add_attribute(E_i, A_k)
If $\exists E_j \cong E_m \mid V \notin M_{Em} \wedge V = Exp(E_j)$
Then If $\exists A_{mk} \in E_m.A \mid E_m.A_{mk} \cong ADP$, where $ADP=E_j....E_i.A_k \vee ADP=(E_i....E_j)^{-1}.A_k$
Then $V.A := \{ADP\} \cup X(E_j)$
add_mapping(V, G_{Em})
search_operation(G_{Em})

Example: Suppose that the operation add_attribute($movie_2$, $year_2$(string,(1,1))) is performed on the local data source S_2. This event triggers Rule 1, Rule 2, Rule 3 and Rule 4, which must be evaluated for both mediation entities $movie_m$ and $actor_m$. However, only the condition of Rule 1($movie_m$) is true. When the rule is executed the first action consists in adding the attribute $year_2$ into the mapping view V_{movie2}. Finally, new operations are searched and added in the graph G_{moviem}; in our example, no new operation can be derived. Figure 6.10 shows the resulting operation graph for the mediation entity $movie_m$.

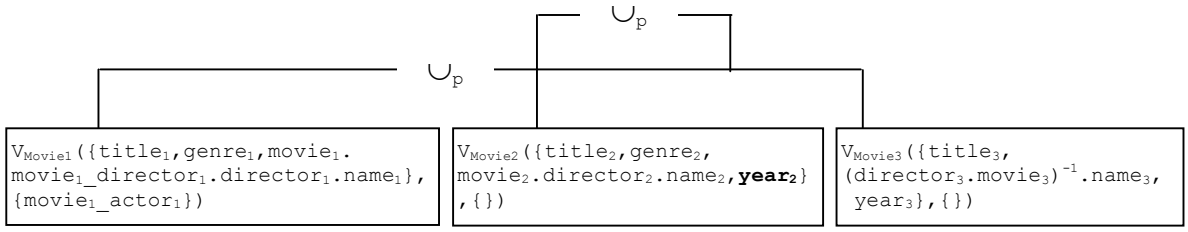

Figure 6.10 – Operation graph G_{moviem} after the propagation of the add_attribute($movie_2$, $year_2$(string, (1,1))) operation

6.3.2 Removing an attribute from a source entity

The following ECA rules (Rule 5 and Rule 6) update the set of mapping views and the set of candidate operations that define the mediation entity E_m after the deletion of the attribute A_k from the source entity E_i, as follows.

- **Rule 5:** checks if there is a mapping view V associated with E_m over E_i and if the removed attribute A_k belongs to the mapping view V. To reflect the deletion of the attribute A_k from the local entity E_i, the attribute A_k must be removed from the mapping view V and all edges in G_{Em} representing operations which are no longer possible must be removed. As defined in Chapter 5, the only restriction on the usage of the set operators between mapping views is that a mapping operator may be applied only when the mapping views have a common identifier which is defined through the mapping attributes. When an attribute A_k is removed from a source entity E_i then it is necessary to verify if there are some mapping operators between the mapping view V, derived from E_i, and other mapping views V′ which depend on the removed attribute (A_k belongs to the set of mapping attributes between E_i and E_x, such that V′ is derived from E_x). In this case, the set operator must be removed.

```
Rule 5(Em)
When remove_attribute(Ei, Ak)
If ∃V ∈ MEm | V = Exp(Ei)
Then If Ak ∈ V.A
     Then V.A := V.A - {Ak}
          remove_operations(GEm, V, Ak)
```

- **Rule 6:** checks if there is a source entity E_j that is semantically equivalent to E_m and if there is a mapping view V associated with E_m over E_j. The condition part of the rule also verifies if there is an attribute derivation path from the entity type E_j to the attribute A_k. In the positive case, this derivation path must be removed from the mapping view V. Then it is necessary to verify if there are some mapping operators between the mapping view V and other mapping views V′ which depend on the removed derivation path and, therefore, must be removed.

Rule 6(E_m)
When remove_attribute(E_i, A_k)
If $\exists E_j \cong E_m \mid \exists V \in M_{Em} \wedge V = Exp(E_j)$
Then If $\exists ADP \in \underline{E_j.A}$, where $ADP = E_j.... .E_i.A_k \vee ADP = (E_i.... .E_j)^{-1}.A_k$
Then $\underline{V.A} := \underline{V.A} - ADP$

Example: Suppose that the operation remove_attribute($movie_3$, $year_3$) is performed on the local data source S_3. This event triggers Rule 5 and Rule 6, which must be evaluated for both mediation entities $movie_m$ and $actor_m$. However, only the condition of Rule 5($movie_m$) is true. When the rule is executed the first action consists in removing the attribute $year_3$ from the mapping view V_{movie3}. Finally, the operations in the graph G_{moviem} are analysed to identify invalid ones; in our example, no operation become invalid. Figure 6.11 shows the resulting operation graph for the mediation entity $movie_m$. Notice that after this modification, the $year_m$ attribute of the mediation $movie_m$ becomes no longer computable.

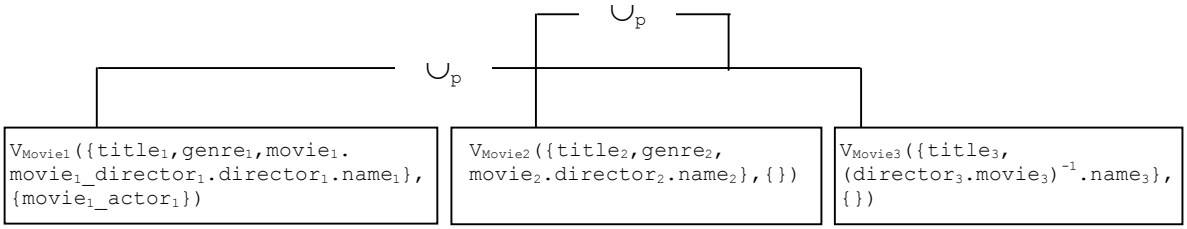

Figure 6.11 – Operation graph G_{moviem} after the propagation of the remove_attribute($movie_3$, $year_3$) operation

6.3.3 Adding an entity into a source schema

The following rules (Rule 7, Rule 8 and Rule 9) update the set of mapping views defining the mediation entity E_m after the addition of a source entity E_i.

- **Rule 7**: the condition part of this rule checks if :

  i) There are attributes $A_1, ..., A_n$ in the source entity E_i that are semantically equivalent to attributes $A_{m1}, ..., A_{mn}$ of the mediation entity E_m,

  ii) There are some attribute derivation paths from E_i to attributes $\{A_1, ..., A_p\}$ which are semantically equivalent to a set of attributes $\{A_{m1}, ..., A_{mo}\}$ of the mediation entity E_m,

  iii) There are containment relationships $R_1, ..., R_n$ in the source entity E_i linking the source entity E_i with a set of subentities $\{E_1, ..., E_t\}$ which are semantically equivalent to a set of subentities $\{E_{m1}, ..., E_{mn}\}$ of the mediation entity E_m,

  iv) There are some entity derivation paths (EDP) from E_i to subentities $\{E_1, ..., E_t\}$ which are semantically equivalent to a set of subentities $\{E_{m1}, ..., E_{mn}\}$ of the mediation entity E_m,

The action part specifies that a mapping view V must be added into the operation graph G_{Em} and the operations between V and the other mapping views in G_{Em} must be identified. The new mapping view will be composed by the attributes, containment relationships and derivation paths identified during the condition evaluation.

Rule 7(E_m)
When add_entity($\underline{E_i}$, S)
If $\exists\{A_1,\ldots,A_n\} \in \underline{E_i.A} \mid \forall t = 1,\ldots,n, \exists\ E_m.A_m \cong E_i.A_t$ **or**

$\exists\{ADP_1,\ldots,ADP_n\}$, where $ADP_t = E_i.\ \ldots\ .A_k \vee ADP_t = (E'.\ \ldots\ .E_i)^{-1}.A_k \mid$
$\forall t = 1,\ldots,n, \exists E_m.A_m \cong ADP_t$ **or**

$\exists\{R_1,\ldots,R_k\} \in \underline{E_i.R} \mid \forall t = 1,\ldots,n, \exists\ E_m.R_m.E_m' \cong E_i.R_t.E'$ **or**

$\exists\{EDP_1,\ldots,EDP_p\}$, where $EDP_t = E_i.\ \ldots\ .E' \vee EDP_t = (E'.\ \ldots\ .E_i)^{-1} \mid$
$\forall t = 1,\ldots,n, \exists E_m.R_m.E_m' \cong EDP_t$

Then $\underline{V.A} := \{A_1,\ldots,A_n\} \cup \{ADP_1,\ \ldots,\ ADP_n\} \cup X(E_i)$
$\underline{V.R} := \{R_1,\ldots,R_p\} \cup \{EDP_1,\ \ldots,EDP_p\}$
add_mapping(V, G_{Em})
search_operation(G_{Em})

- **Rule 8**: checks if there is a source entity E_j that is semantically equivalent to E_m and if there is a mapping view V associated with E_m over E_j. In this case, the condition part of the rule also verifies if there are new derivation paths from the entity type E_j that can be computed after the insertion of the entity type E_i as follows:

  i) If there are some attribute derivation paths from E_j to attributes $\{A_1,\ldots,A_p\}$ which are semantically equivalent to a set of attributes $\{A_{m1},\ldots,A_{mo}\}$ of the mediation entity E_m,

  ii) There are some entity derivation paths from E_j to subentities $\{E_1,\ldots,E_t\}$ which are semantically equivalent to a set of subentities entities $\{E_{m1},\ldots,E_{mn}\}$ of the mediation entity E_m,

  Then these derivations paths (attribute derivation path or entity derivation path) must be added into the mapping view V.

Rule 8(E_m)
When add_entity($\underline{E_i}$, S)
If $\exists E_j \cong E_m \mid \exists V \in M_{Em} \wedge V = Exp(E_j)$
Then If $\exists\{ADP_1, \ldots, ADP_n\}$, where $ADP_t = (E'.\ldots.E_i.R_k. \ldots E_j)^{-1}.A_k \mid$
$\forall t = 1,\ldots,n, \exists E_m.A_m \cong ADP_t$ **or**
$\exists\{EDP_1, \ldots, EDP_p\}$, where $EDP_t = (E'.\ldots.E_i.R_k. \ldots .E_j)^{-1} \mid$
$\forall t = 1,\ldots,p, \exists E_m.R_m.E_m' \cong EDP_t$
Then $\underline{V.A} := \underline{V.A} \cup \{ADP_1,\ldots,ADP_n\}$
$\underline{V.R} := \underline{V.R} \cup \{EDP_1,\ldots,EDP_p\}$

- **Rule 9**: is similar to Rule 8, however the condition part checks if there is no mapping view V associated with E_m over E_j. In this case a new mapping view V must be created whose content will be composed by the entity derivation paths and attribute derivation paths identified during the condition evaluation. V must be added into the operation graph G_{Em} and the operations between the new mapping view and the other mapping views in G_{Em} must be identified.

Rule 9(E_m)
When add_entity($\underline{E_i}$, S)
If $\exists E_j \cong E_m \mid V \notin M_{Em} \wedge V = Exp(E_j)$
Then If $\exists\{ADP_1, \ldots, ADP_n\}$, where $ADP_t = (E'.\ldots.E_i.R_k. \ldots E_j)^{-1}.A_k \mid$
$\forall t = 1,\ldots,n, \exists E_m.A_m \cong ADP_t$ **or**
$\exists\{EDP_1, \ldots, EDP_p\}$, where $EDP_t = (E'.\ldots.E_i.R_k. \ldots .E_j)^{-1} \mid$
$\forall t = 1,\ldots,p, \exists E_m.R_m.E_m' \cong EDP_t$
Then $\underline{V.A} := X(E_j) \cup \{ADP_1,\ldots,ADP_n\}$
$\underline{V.R} := \{EDP_1,\ldots,EDP_p\}$
add_mapping(V, G_{Em})
search_operation(G_{Em})

Example: Suppose that the operation add_entity($actor_3$({$name_3$,$biography_3$},{}), S_3) is performed on the local data source S_3. This event triggers Rule 7, Rule 8 and Rule 9, which must be evaluated for both mediation entities $movie_m$ and $actor_m$. In this case, Rule 7 is evaluted as true only for $actor_m$. When the rule is executed a new mapping view derived from $actor_3$ is created (V_{Actor3}({$name_3$},{})) and it is added in the graph G_{actorm}. Next, new operations are searched and added into the graph G_{moviem}, based on the correspondence assertion $actor_1 \cap actor_3 \neq \emptyset$. Figure 6.12 shows the resulting operation graph for the mediation entity $actor_m$. Notice that this operation graph shows all operations that must be performed between V_{Actor1} and V_{Actor3}. Each edge on the graph is labeled with one of the possible mapping operators to be applied between V_{Actor1} and V_{Actor3}.Since there is an intersection between the instances of V_{Actor1} and V_{Actor3}, then all mapping operators should be applied between them. During the generation of the mediation query Q($actor_m$), one of these operators will be chosen to define the computing expression of $actor_m$.

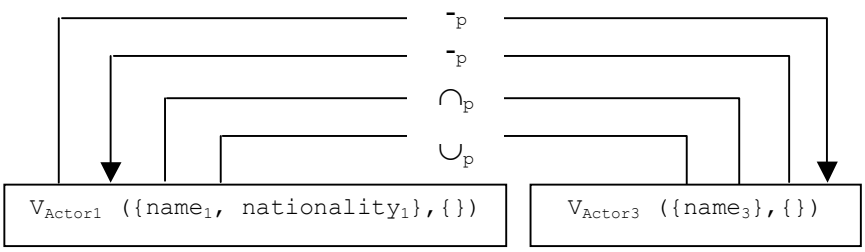

Figure 6.12 - Operation graph G_{actorm} after the propagation of the operation add_entity(actor$_3$({name$_3$,biography$_3$},{actor$_3$_movie$_3$}))

6.3.4 Removing an entity from a source schema

The following rules (Rule 10 and Rule 11) update the set of mapping views associated with the entity E_m in the mediation schema after the deletion of a source entity E_i.

- **Rule 10**: the condition part of this rule checks if there is a mapping view V associated with E_m over E_i. To reflect the deletion of the local entity E_i, the corresponding mapping view V must be removed from the operation graph G_{Em}, along with all the operations involving the mapping view V.

Rule 10(E_m)
When remove_entity(E_i, S)
If $\exists V \in M_{Em} \mid V = Exp(E_i)$
Then remove_mapping(V, G_{Em})

- **Rule 11**: checks if there is a source entity E_j that is semantically equivalent to E_m and if there is a mapping view V associated with E_m over E_j. In this case, the condition part of the rule also verifies if there are derivation paths from the entity type E_j that can not be computed after the deletion of E_i. In this case, these derivation paths (attribute derivation path or entity derivation path) must be removed from the mapping view V. Then, it is necessary to verify if there are some mapping operators between the mapping view V and other mapping views V' which depend on one of the removed attribute derivation paths. In this case, the mapping operator must be removed.

Rule 11(E_m)
When remove_entity(E_i, S)
If $\exists E_j \cong E_m \mid \exists V \in M_{Em} \wedge V = Exp(E_j)$
Then If $\exists \{ADP_1, \ldots, ADP_n\}$, where $ADP_t = (E'.\ldots.E_i.R_k. \ldots E_j)^{-1}.A_k \mid$
$\forall t = 1,\ldots,n, \exists E_m.A_m \cong ADP_t$ **or**
$\exists \{EDP_1, \ldots, EDP_p\}$, where $EDP_t = (E'.\ldots.E_i.R_k. \ldots .E_j)^{-1} \mid$
$\forall t = 1,\ldots,p, \exists E_m.R_m.E_m' \cong EDP_t$
Then V.A := V.A - $\{ADP_1,\ldots,ADP_n\}$
V.R := V.R - $\{EDP_1,\ldots,EDP_p\}$
remove_operations(G_{Em}, V, $\{ADP_1,\ldots,ADP_n\}$)

Example: Suppose that the operation remove_entity($movie_2$, S_2) is performed on the local data source S_2. This operation is followed by the operation remove_entity ($director_2$, S_2). Each one of these events must be propagated separately. The event remove_entity($movie_2$, S_2) triggers Rule 10 which is evaluated as true only for $movie_m$. When the rule is executed the only action consists in removing the mapping view V_{Movie2} from the operation graph G_{moviem} and all the operations associated with V_{Movie2} (the union operator between V_{Movie2} and V_{Movie3}). Figure 6.13 shows the resulting operation graph for the mediation entity $movie_m$. The event remove_entity($director_2$, S_2) also triggers Rule 10. However, this rule is not evaluated as true neither for $movie_m$ nor for $actor_m$, because there are no mapping views in the operation graphs G_{moviem} and G_{actorm} that are either derived from the source entity $director_2$ or have a derivation path that can not be computed after the removal of $director_2$.

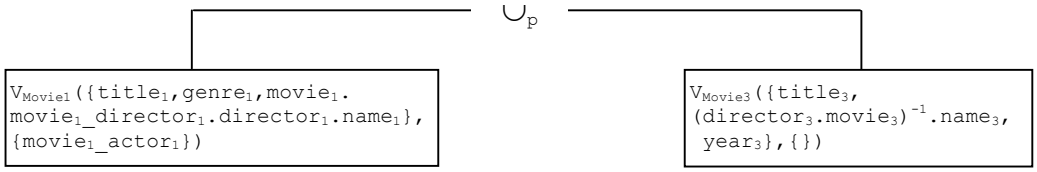

Figure 6.13 - Operation graph G_{movie} after the propagation of the operation remove_entity($movie_2$, S_2)

6.3.5 Adding a containment relationship into a source entity

The following ECA rules (Rule 12, Rule 13, Rule 14 and Rule 15) update the mapping views corresponding to the mediation entity E_m after an insertion of a new containment relationship R_k into a source entity E_i.

- **Rule 12:** the condition part of this rule checks if there is a mapping view V associated with E_m over E_i. In the positive case, the following conditions are also verified:

  i) if the relatiosnhip R_k links the source entity E_i with a subentity E' which is semantically equivalent to a subentity E_m' of the mediation entity E_m,

  ii) if there are some entity derivation paths from E_i to subentities $\{E_1, ..., E_t\}$, where E_i is semantically equivalent to a subentity E_m' of the mediation entity E_m, such that R_k participates in the derivation path,

  iii) if there are some attribute derivation paths from E_i to attributes $\{A_1, ..., A_p\}$, where A_x is semantically equivalent to an attribute A_m of the mediation entity E_m, such that R_k participates in the derivation path.

If the first condition is true then the relationship R_k must be inserted into the set of containment relationships of the mapping view V. If the second condition is true, then the entity derivation paths must be inserted into the set of containment relationships of the mapping view V. When the third condition is true, the attribute derivation paths must be inserted into the set of attributes of the mapping view V.

Rule 12 (E_m)
When add_contains_rel(E_i, R_k)
If $\exists\ V \in M_{Em} \mid V = Exp(E_i)$
Then If $\exists\ R_{mk} \in \underline{R_m.R} \mid E_m.R_m.E_m' \cong E_i.R_k.E'$ **or**
$\exists\{EDP_1, \ldots, EDP_p\}$, where $EDP_t = E_i.R_k. \ldots .E'$
$\mid \forall t = 1,\ldots,p,\ \exists E_m.R_m.E_m' \cong EDP_t$ **or**
$\exists\{ADP_1, \ldots, ADP_n\}$, where $ADP_t = E_i.R_k. \ldots .A_x \mid \forall t = 1,\ldots,n,\ \exists E_m.A_m \cong ADP_t$
Then $\underline{V.A} := \underline{V.A} \cup \{ADP_1, \ldots, ADP_n\}$
$\underline{V.R} := \underline{V.R} \cup \{R_k\} \cup \{EDP_1, \ldots, EDP_p\}$

- **Rule 13**: is similar to Rule 12, however the condition part checks if there is no mapping view V associated with E_m over E_i. In this case a new mapping view V must be created whose content will be composed by the containment relationships, entity derivation paths and attribute derivation paths identified during the condition evaluation. V must be added into the operation graph G_{Em} and the operations between V and the other mapping views in G_{Em} must be identified.

Rule 13(E_m)
When add_contains_rel(E_i, R_k)
If $V \notin M_{Em} \mid V = Exp(E_i)$
Then If $\exists E_m.R_{mk} \in \underline{E_m.R} \mid E_m.R_m.E_m' \cong E_i.R_k.E'$ **or**
$\exists\{EDP_1, \ldots, EDP_p\}$, where $EDP_t = E_i.R_k. \ldots .E'$
$\mid \forall t = 1,\ldots,p,\ \exists E_m.R_m.E_m' \cong EDP_t$ **or**
$\exists\{ADP_1, \ldots, ADP_n\}$, where $ADP_t = E_i.R_k. \ldots .A_k$
$\mid \forall t = 1,\ldots,n,\ \exists E_m.A_m \cong ADP_t$
Then $\underline{V.A} := \{X\} \cup \{ADP_1, \ldots, ADP_n\}$
$\underline{V.R} := \{R_k\} \cup \{EDP_1, \ldots, EDP_p\}$
add_mapping(V, G_{Em})
search_operation(G_{Em})

- **Rule 14**: checks if there is a source entity E_j that is semantically equivalent to E_m and if there is a mapping view V associated with E_m over E_j. In this case, the condition part of the rule also verifies if there are new derivation paths from the entity type E_j that become computable after the insertion of the relationship R_k as follows:

i) If there are some attribute derivation paths from E_j to attributes $\{A_1,...,A_p\}$ which are semantically equivalent to a set of attributes $\{A_{m1},...,A_{mo}\}$ of the mediation entity E_m, such that R_k participates in the derivation path,

ii) There are some entity derivation paths from E_j to subentities $\{E_1,...,E_t\}$ which are semantically equivalent to a set of subentities entities $\{E_{m1},...,E_{mn}\}$ of the mediation entity E_m, such that R_k participates in the derivation path.

Then these derivation paths must be added into the mapping view V.

Rule 14(E_m)
When add_contains_rel(E_i, R_k)
If $\exists E_j \cong E_m \mid \exists V \in M_{Em} \wedge V = Exp(E_j)$
Then If $\exists\{ADP_1, ...,ADP_n\}$, where $ADP_t = (E'....E_i.R_k. \ldots E_j)^{-1}.A_x \mid$
$\forall t = 1,...,n, \exists E_m.A_m \cong ADP_t$ **or**
$\exists\{EDP_1, ...,EDP_p\}$, where $EDP_t = (E'....E_i.R_k. \ldots .E_j)^{-1} \mid$
$\forall t = 1,...,p, \exists E_m.R_m.E_m' \cong EDP_t$
Then $\underline{V.A} := \underline{V.A} \cup \{ADP_1,...,ADP_n\}$
$\underline{V.R} := \underline{V.R} \cup \{EDP_1,...,EDP_p\}$

- **Rule 15**: is similar to Rule 14, however the condition part checks if there is no mapping view V associated with E_m over E_j. In this case a new mapping view V must be created whose content will be composed by the entity derivation paths and attribute derivation paths identified during the condition evaluation. V must be added into the operation graph G_{Em} and the operations between V and the other mapping views in G_{Em} must be identified.

Rule 15(E_m)
When add_contains_rel(E_i, R_k)
If $\exists E_j \cong E_m \mid V \notin M_{Em} \wedge V = Exp(E_j)$
Then If $\exists\{ADP_1, ...,ADP_n\}$, where $ADP_t = (E'....E_i.R_k. \ldots E_j)^{-1}.A_x \mid$
$\forall t = 1,...,n, \exists E_m.A_m \cong ADP_t$ **or**
$\exists\{EDP_1, ...,EDP_p\}$, where $EDP_t = (E'....E_i.R_k. \ldots .E_j)^{-1} \mid$
$\forall t = 1,...,p, \exists E_m.R_m.E_m' \cong EDP_t$
Then $\underline{V.A} := X(E_j) \cup \{ADP_1,...,ADP_n\}$
$\underline{V.R} := \{EDP_1,...,EDP_p\}$
add_mapping(V, G_{Em})
search_operation(G_{Em})

Example: Suppose that after the addition of the entity type $actor_3$ in the source S_3 we want to add a containment relationship between $actor_3$ and $movie_3$. This is done through the operation: add_contains_rel($actor_3$,$actor_3$_$movie_3$). This event triggers Rule 12, Rule 13, Rule 14 and Rule 15, which must be evaluted for both $movie_m$ and $actor_m$. However, only the Rule 14 is evaluated as true for $movie_m$. When the rule is executed the entity

derivation path $(movie_3.movie_3\_actor_3.actor_3)^{-1}$ is inserted in the set of relationships of the mapping view V_{Movie3}. Figure 6.14 shows the resulting operation graph for the mediation entity $movie_m$.

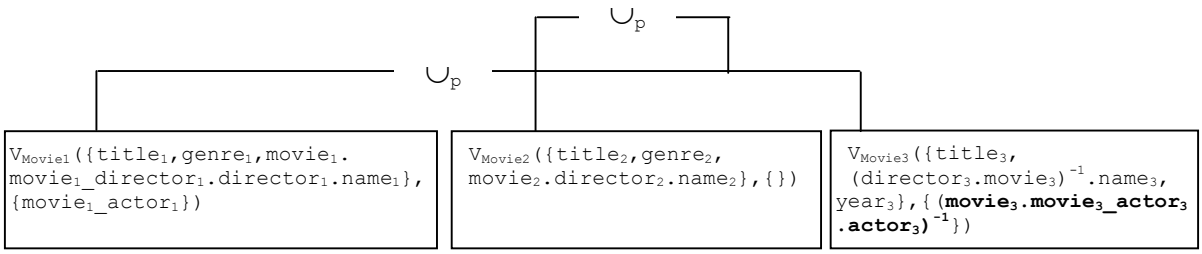

Figure 6.14 - Operation graph G_{movie} after the propagation of the operation add_contains_relationship($actor_3$,$actor_3$_$movie_3$)

6.3.6 Removing a containment relationship from a source entity

The following ECA rules (Rule 16 and Rule 17) update the mapping views corresponding to the mediation entity E_m after the deletion of the containment relationship R_k from the source entity E_i.

- **Rule 16**: the condition part of this rule checks if there is a mapping view V associated with E_m over E_i. In the positive case, the following conditions are also verified:

  i) if R_k belongs to the set of containment relationships of the mapping view V,

  ii) if there are attribute derivation paths or entity derivation paths in the mapping view V that depends on the link between E_i and R_k.

  Therefore, R_k must be removed from the set of containment relationships of the entity V along with the entity derivation paths identified during the condition evaluation. Besides, the attribute derivation paths identified during the condition evaluation must be removed from the set of attributes of V. Then it is necessary to verify if there are some mapping operators between the mapping view V and other mapping views V' which depend on the removed attribute derivation paths. In this case, the mapping operator must be removed from G_{EM}.

Rule 16(E_m)
When remove_contains_rel(E_i, R_k)
If $\exists V \in M_{Em} \mid V = Exp(E_i)$

Then If $R_k \in \underline{V.R}$ **or**

$\exists\{ADP_1, \ldots, ADP_n\}$, where $ADP_t = E_i.R_k. \ldots .A_k \wedge$
$\forall t = 1,\ldots,n,\ ADP_t \in \underline{V.A}$ **or**

$\exists\{EDP_1, \ldots, EDP_p\}$, where $EDP_t = E_i.R_k. \ldots .E' \wedge$
$\forall t = 1,\ldots,p,\ EDP_t \in \underline{V.R}$

Then $\underline{V.R} := \underline{V.R} - \{R_k\} - \{EDP_1,\ldots,EDP_p\}$
$\underline{V.A} := \underline{V.A} - \{ADP_1,\ldots,ADP_n\}$
remove_operations(G_{Em}, V, R_k)

- **Rule 17**: checks if there is a source entity E_j that is semantically equivalent to E_m and if there is a mapping view V associated with E_m over E_j. In this case, the condition part of the rule also verifies if there are derivation paths from the entity type E_j that belong to V and that can not be computed after the removal of the relationship R_k. In this case, these derivation paths (attribute derivation path or entity derivation path) must be removed from the mapping view V. Then, it is necessary to verify if there are some mapping operators between the mapping view V and other mapping views V' which depend on the removed derivation paths. In this case, the mapping operator must be removed.

Rule 17(E_m)
When remove_contains_rel(E_i, R_k)
If $\exists E_j \cong E_m \mid \exists V \in M_{Em} \wedge V = Exp(E_j)$

Then If $\exists\{ADP_1,\ldots,ADP_n\}$, where $ADP_t = E_j. \ldots .E_i.R_k. \ldots E'.A_k \vee$
$ADP_t = (E'. \ldots .E_i.R_k. \ldots E_j)^{-1}.A_k \wedge \forall t = 1,\ldots,n,\ ADP_t \in \underline{V.A}$ **or**

$\exists\{EDP_1, \ldots, EDP_p\}$, where $EDP_t = E_j. \ldots .E_i.R_k. \ldots .E'' \vee$
$EDP_t = (E'. \ldots .E_i.R_k. \ldots .E_j)^{-1} \wedge \forall t = 1,\ldots,p,\ EDP_t \in \underline{V.R}$

Then $\underline{V.R} := \underline{V.R} - \{EDP_1,\ldots,EDP_p\}$
$\underline{V.A} := \underline{V.A} - \{ADP_1,\ldots,ADP_n\}$
remove_operations(G_{Em}, V, R_k)

Example: Suppose that the operation remove_contains_rel($director_3$, $director\_movie_3$) is performed on the local data source S_3. This event triggers Rule 16 and Rule 17. However, only the Rule 17 is evaluated as true for $movie_m$. When the rule is executed the path $(director_3.movie_3)^{-1}.name_3$ is removed from the set of attributes of the mapping view V_{movie3}. Figure 6.15 shows the resulting operation graph for the mediation entity $movie_m$.

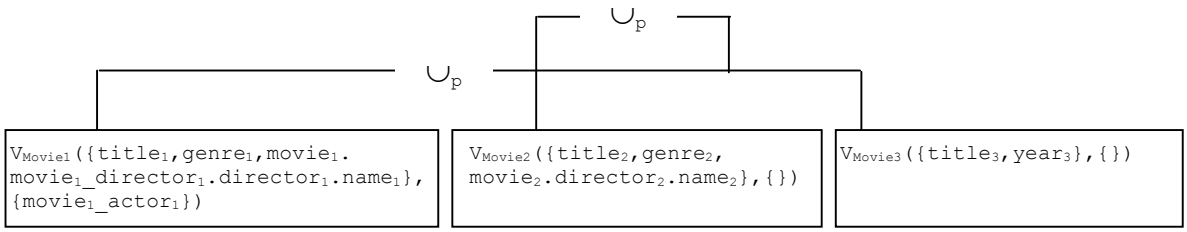

Figure 6.15 - Operation graph G_{movie} after the propagation of the operation remove_contains_relationship(director$_3$, director_movie$_3$)

Once the data source schemas changes are propagated in the set of mapping views and the set of candidate operations describing each mediation entity, and if these sets have been modified, the associated mediation query may become invalid and a new mediation query has to be generated. The propagation of data source schemas changes in the mapping views and the corresponding mediation queries generation are the two main tasks of the source schemas evolution process which is described in section 6.5.

6.4 Using mapping views evolution rules to propagate users' requirements changes

The *Users' Requirements Manager* receives users' needs modifications and identifies the change operations to be performed in the mediation schema in order to reproduce them. If these changes can be reflected in the mediation queries, the modifications on the mediation schema are committed; otherwise the user is informed that his or her new requirements cannot be satisfied. Remember that the problem of propagating users' requirements into the mediation queries consists, first in propagating these changes into the mapping views, and second in modifying the mediation queries in order to take into account the modifications in the set of mapping views.

The propagation of users' requirements changes into mapping views is done through a set of mapping views evolution rules, which identifies information from the source entities relevant to the computation of the new requirements and performs the necessary modifications into the set of mapping views.

In the following, the mapping views evolution rules are classified according to the type of mediation schema change operation. Depending on the operation, the rules are evaluated over the whole set of source entities or over the mapping views associated with the mediation entity that is being modified. Each rule has a name and a parameter denoted `V` which represents a mapping view or a parameter `E`$_i$ which represents a source entity. Next sections use as example

the mediation schema and the source schemas introduced in section 5.3 and reviewed in section 6.3.

6.4.1 Adding an attribute into a mediation entity

The following ECA rules (Rule 18 and Rule 19) update the mapping views corresponding to the mediation entity E_m after an insertion of a new attribute A_m into E_m.

- **Rule 18:** for each mapping view V derived from a source entity E_i and associated with the mediation entity E_m, this rule investigates the set of attributes of E_i to check if there is an attribute A_k semantically equivalent to the new attribute A_m. If the condition is true, the attribute A_k must be inserted into the set of attributes of the mapping view V and new operations must be searched in G_{EM}.

Rule 18(V)
When add_attribute(E_m, $\underline{A_m}$)
If $\exists A_k \in \underline{E_i.A}$, where $V = Exp(E_i)$ | $A_k \notin \underline{V.A} \wedge E_m.A_m \cong E_i.A_k$
Then $\underline{V.A} := \underline{V.A} \cup A_k$
search_operation(G_{Em})

- **Rule 19:** for each mapping view V derived from a source entity E_i and associated with the mediation entity E_m, this rule checks if there is an attribute derivation from the source entity E_i to an attribute A_k semantically equivalent to the new attribute A_m. If the condition is true, the attribute derivation path must be inserted into the set of attributes of the mapping view V.

Rule 19(V)
When add_attribute(E_m, $\underline{A_m}$)
If $\exists ADP$, $ADP = E_i. \ldots .A_k \vee ADP = (E'. \ldots .E_i)^{-1}.A_k$, where $V = Exp(E_i)$ |
$ADP \notin \underline{V.A} \wedge E_m.A_m \cong ADP$
Then $\underline{V.A} := \underline{V.A} \cup ADP$

Example: Suppose that the operation add_attribute($movie_m$, $duration_m$(string,(0,1))) is performed on the mediation schema S_{med}. This event triggers Rule 18 and Rule 19, which must be evaluted for both V_{movie1}, V_{movie2} and V_{movie3}. However only the $movie_1$ entity has an attribute, called $duration_1$, that is semantically equivalent to the $duration_m$ attribute. So, $duration_1$ must be inserted in the set of attributes of V_{movie1}. Figure 6.16 shows the resulting operation graph for the mediation entity $movie_m$.

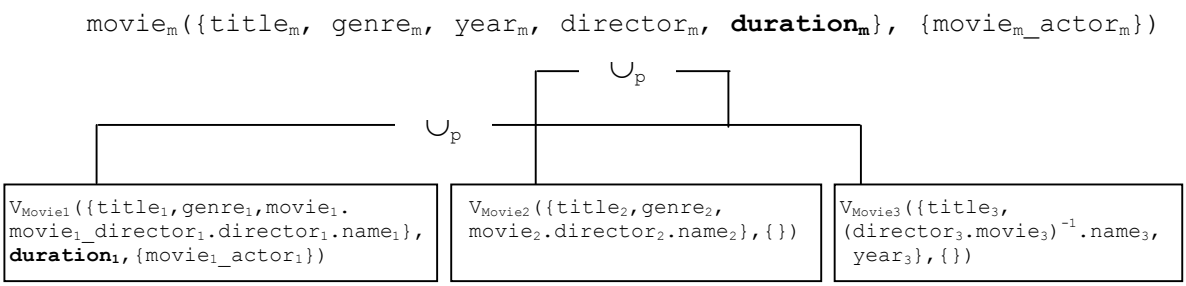

Figure 6.16 - Operation graph G_{movie} after the propagation of the operation add_attribute(movie$_m$, duration$_m$(string, (0,1)))

6.4.2 Removing an attribute from a mediation entity

The following ECA rules (Rule 20 and Rule 21) update the mapping views corresponding to the mediation entity E_m after the deletion of the attribute A_m from E_m.

- **Rule 20:** for each mapping view V derived from a source entity E_i and associated with the mediation entity E_m, this rule investigates the set of attributes of E_i to check if there is an attribute A_k semantically equivalent to A_m. If the condition is true, the attribute A_k must be removed from the set of attributes of the mapping view V. Then it is necessary to verify if there are some mapping operators between the mapping view V and other mapping views V′ which depend on the removed attribute and must be removed from G_{EM}.

Rule 20(V)
When remove_attribute(E_m, A_m)
If $\exists A_k \in$ V.A $|\ E_m.A_m \cong E_i.A_k$, where $V = Exp(E_i)$
Then V.A := V.A - A_k
remove_operations(G_{Em}, V, A_k)

- **Rule 21:** for each mapping view V associated with the mediation entity E_m, this rule checks if there is an attribute derivation path, in the set of attributes of the mapping view V, semantically equivalent to the attribute A_m. If the condition is true, the attribute derivation path must be removed from the set of attributes of the mapping view V. Then it is necessary to verify if there are some mapping operators between the mapping view V and other mapping views V′ which depend on the removed attribute derivation path. In this case, the mapping operator must be removed from G_{EM}.

Rule 21(V)
When remove_attribute(E_m, A_m)
If $\exists ADP = E_i. \ldots .A_k \vee ADP = (E'. \ldots .E_i)^{-1}.A_k |\ E_m.A_m \cong ADP$
Then V.A := V.A - ADP
remove_operations(G_{Em}, V, ADP)

Example: Suppose that the operation remove_attribute($movie_m$, $genre_m$) is performed in the mediation schema S_{med}. This event triggers Rule 20 and Rule 21, which must be evaluted for both V_{movie1}, V_{movie2} and V_{movie3}. In this case, V_{movie1} and V_{movie2} have the attributes $genre_1$ and $genre_2$, which are semantically equivalent to the $genre_m$ attribute. So, $genre_1$ and $genre_2$ must be removed from the set of attributes of V_{movie1} and V_{movie2}, respectively. Figure 6.17 shows the resulting operation graph for the mediation entity $movie_m$.

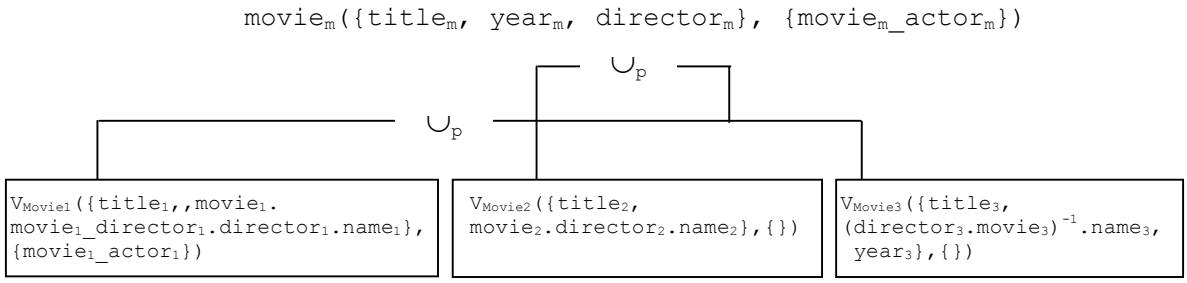

Figure 6.17 - Operation graph G_{movie} after the propagation of the operation remove_attribute($movie_m$, $genre_m$)

6.4.3 Adding an entity into the mediation schema

The following ECA rule (Rule 22) identifies the mapping views relevant to compute a mediation entity E_m.

- **Rule 22:** identifies the source entities relevant to compute the new mediation entity E_m. This rule is evaluated for each source entity E_i. If E_i is semantically equivalent to E_m then the attributes, containment relationships and derivation paths of E_i, which are considered relevant to compute E_m, are identified. Such elements will compose the content of the new mapping view V, which will be inserted into the operation graph G_{Em}.

Rule 22(E_i)
When add_entity(E_m, S_m)
If $E_i \in S.E \cong E_m$
Then if $\exists\{A_1,\ldots,A_n\} \in E_i.A \mid \forall t = 1,\ldots,n, \exists E_m.A_m \cong E_i.A_t$ **or**
$\exists\{ADP_1,\ldots,ADP_n\}$, where $ADP_t = E_i.\ldots.A_k \vee ADP_t = (E'.\ldots.E_i)^{-1}.A_k \mid$
$\forall t = 1,\ldots,n, \exists E_m.A_m \cong ADP_t$ **or**
$\exists\{R_1,\ldots,R_k\} \in E_i.R \mid \forall t = 1,\ldots,n, \exists E_m.R_m.E_m' \cong E_i.R_t.E'$ **or**
$\exists\{EDP_1,\ldots,EDP_p\}$, where $EDP_t = E_i.\ldots.E' \vee EDP_t = (E'.\ldots.E_i)^{-1} \mid$
$\forall t = 1,\ldots,n, \exists E_m.R_m.E_m' \cong EDP_t$
Then $V_{Ei}.A := \{A_1,\ldots,A_n\} \cup \{ADP_1,\ldots,ADP_n\} \cup X(E_i)$
$V_{Ei}.R := \{R_1,\ldots,R_p\} \cup \{EDP_1,\ldots,EDP_p\}$
add_mapping(V_{Ei}, G_{Em})
search_operation(G_{Em})

Example: Suppose that we want to obtain information about the movies' awards. To get this information a new entity type, called $award_m$, must be added into the mediation schema. This is done performing the following change operation: add_entity($award_m$({$category_m$, $year_m$},{}), S_{med}). This operation triggers Rule 22, which is evaluated for all source entities available in the data sources S_1, S_2 and S_3. Since there is no source entity semantically equivalent to $award_m$ then this mediation entity can not be computed and, therefore, the add_entity operation is not commited.

6.4.4 Removing an entity from the mediation schema

When a mediation entity E_m is removed from the mediation schema the mapping views associated with the E_m must be removed. This is done through the following ECA rule.

Rule 23(V_{Ei})
When remove_entity(E_m, S_m)
If $E_i \cong E_m$
Then remove_mapping(G_{Em}, V_{Ei})

Example: Suppose that we want to remove the information about movie's actors from the mediation schema. Therefore, the $actor_m$ entity type must be removed from S_m. This is done performing the following change operation: remove_entity($actor_m$, S_m), which triggers Rule 23. This rule is evaluated only for the mapping view V_{Actor1}. The result of applying Rule 23 over V_{Actor1} is the deletion of V_{Actor1}.

6.4.5 Adding a containment relationship into a mediation entity

The following ECA rules (Rule 24 and Rule 25) update the mapping views corresponding to the mediation entity E_m after an insertion of a containment relationship R_m into E_m.

- **Rule 24:** for each mapping view V derived from E_i and associated with the mediation entity E_m, this rule investigates the set of relationships of E_i, to check if there is a containment relationship R_k, which links the source entity E_i with a subentity E′ such that $E_m.R_m.E_m' \cong E_i.R_k.E'$. If the condition is true, the containment relationship R_k must be inserted into the set of relationships of the mapping view V.

Rule 24(V)
When add_contains_rel(E_m, R_m)
If $\exists R_k \in \underline{E_i.R}$, where V = Exp($E_j$) | $E_m.R_m.E_m' \cong E_{ij}.R_k.E'$
Then $\underline{V.R} := \underline{V.R} \cup R_k$

- **Rule 25:** for each mapping view V associated with the mediation entity E_m, this rule checks if there is an entity derivation path from the source entity E_j, which links the source entity E_i with a subentity E' such that $E_m.R_m.E_m' \cong E_i.\ldots.E'$. E_i is the source entity from which V was derived. If the condition is true, the derivation path must be inserted into the set of relationships of the mapping view V.

Rule 25(V)
When add_contains_rel(E_m, R_m)
If $\exists EDP,\ EDP = E_i.\ \ldots\ .E' \vee EDP_t = (E'.\ \ldots\ .E_i)^{-1}$, where $V = Exp(E_j)$ |
$E_m.R_m.E_m' \cong EDP$
Then $V.R := V.R \cup EDP$

6.4.6 Removing a containment relationship from a mediation entity

The following ECA rules (Rule 26 and Rule 27) update the mapping views corresponding to the mediation entity E_m after the deletion of the containment relationship R_m from E_m.

- **Rule 26:** for each mapping view V associated with the mediation entity E_m, this rule checks if there is a containment relationships R_k, in the set of relationships of the mapping view V, which links the source entity E_{ij} with a subentity E', such that $E_m.R_m.E_m' \cong E_{ij}.R_k.E'$. If the condition is true, the relationship R_k must be removed from the set of relationships of the mapping view V.

Rule 26(V)
When remove_contains_rel(E_m, R_m)
If $\exists R_k \in V.A$ | $E_m.R_m.E_m' \cong E_{ij}.R_k.E'$, where $V = Exp(E_{dj})$
Then $V.R := V.R - R_k$

- **Rule 27:** for each mapping view V associated with the mediation entity E_m, this rule checks if there is a derivation path from the source entity E_{ij}, which links the source entity E_{ij} with a subentity E' such that $E_m.R_m.E_m' \cong E_{ij}.\ldots.E'$. If the condition is true, the derivation path must be removed from the set of relationships of the mapping view V.

Rule 27(V)
When remove_contains_rel(E_m, R_m)
If $\exists EDP = E_{ij}.\ \ldots\ .A_k \vee ADP = (E'.\ \ldots\ .E_{ij})^{-1}.A_k$, where $V = Exp(E_{dj})$ |
$E_m.A_m \cong ADP$
Then $V.A := V.A - ADP$
remove_operations(G_{Em}, V, ADP)

Example: Suppose that we want to remove the information about movie's actors from the

$movie_m$ entity type. Therefore, the containment relationship $movie_m\_actor_m$ must be removed from the mediation entity $movie_m$. This is done performing the following change operation: remove_contains_rel($movie_m$, $movie_m\_actor_m$) . This operation triggers Rule 26 and Rule 27, which must be evaluated for the mapping views $E_{pMovie1}$, $E_{pMovie2}$ and $E_{pMovie3}$. The only combination evaluated as true is Rule 26 applied over the mapping view $E_{pMovie1}$. To reflect the modification done in the mediation schema, the containment relationship $movie_1\_actor_1$ must be removed from the set of relationships of $E_{pMovie1}$. Figure 6.18 shows the resulting operation graph for the mediation entity $movie_m$.

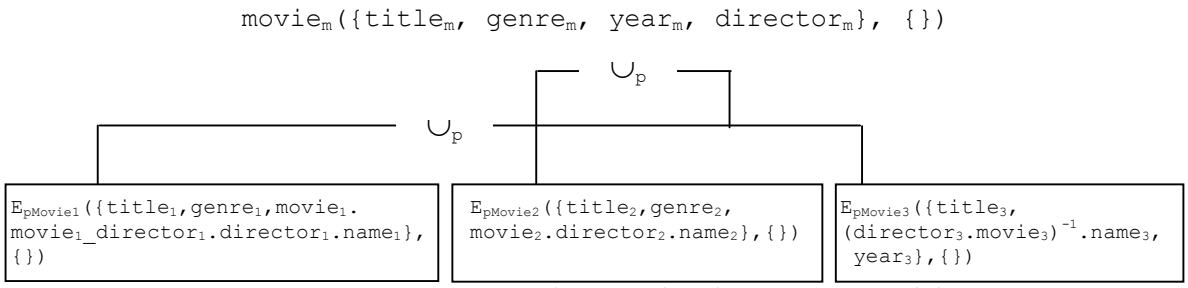

Figure 6.18 - Operation graph G_{movie} after the propagation of the operation remove_contains_rel($movie_m$, $movie_m\_actor_m$)

In the following sections, we will describe the process of propagating data source schemas changes and the process of users' requirements changes propagation.

6.5 The data source schemas changes propagation process

This section describes the process of propagating data source schemas changes (Figure 6.19) to the mediation level. In this process, a change event is a data source schema change represented by one of the schema change operations described in section 6.2. (addition of entities, attributes or containment relationships and removal of entities, attributes or containment relationships). We consider that data source schemas changes are detected by the *Conceptual Schema Manager* (cf. Chapter 3) through the comparison of two different versions of the same exported schema. The *Lookup* module periodically communicates with the data sources in order to extract new versions of their exported schemas.

This propagation process can be divided into two main tasks: mapping views evolution and mediation queries generation, which will be described in the following.

- *Mapping views evolution*

The mapping views evolution consists in propagating data source schemas changes to the

mapping views associated with each entity of the mediation schema using the set of ECA rules described in section 6.3. This task is executed by a set of processes, where each process, called mapping views evolution process and denoted P1, is associated with a distinct data source S_i and is responsible for updating the mapping views derived from source entities belonging to S_i. In this way, different processes can be executed concurrently updating distinct sets of mapping views.

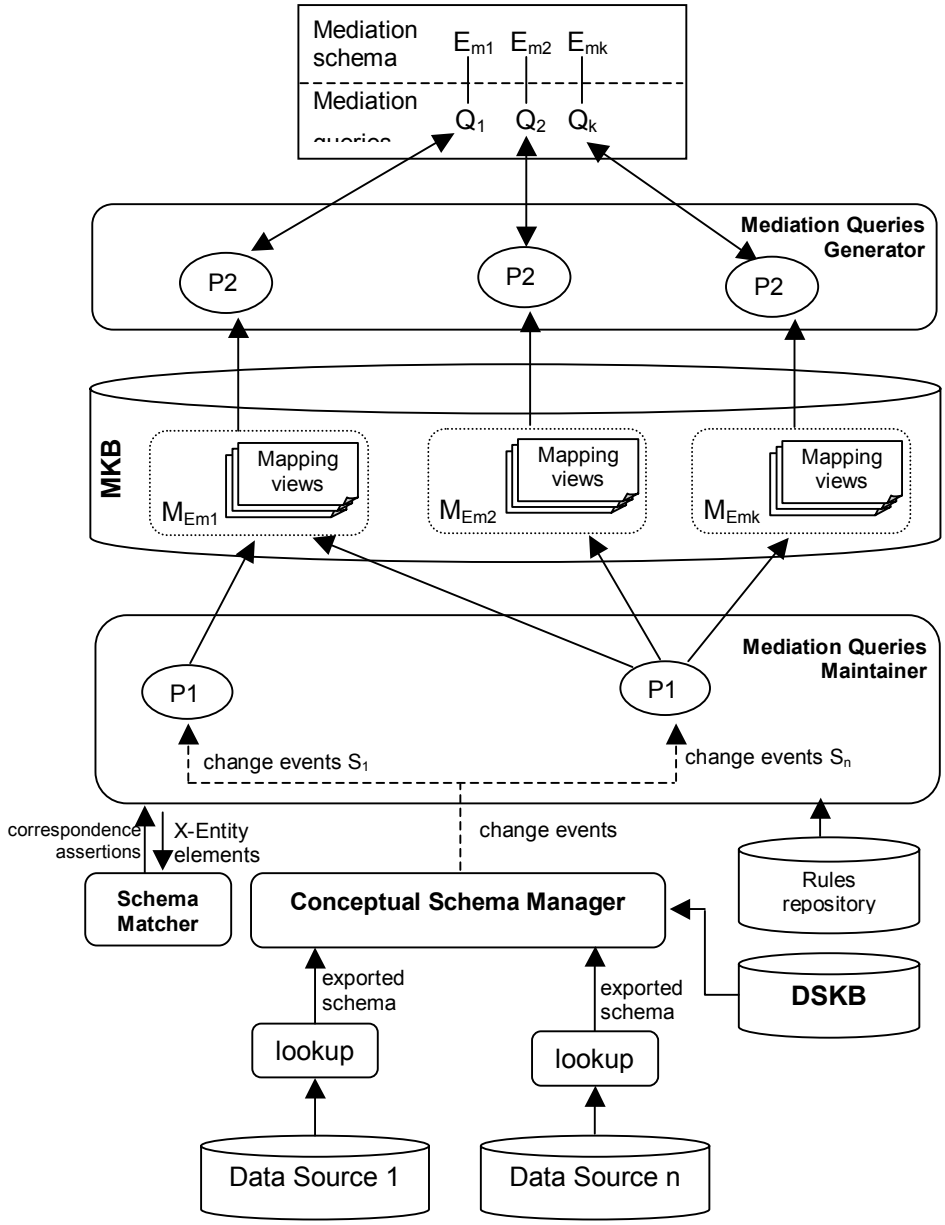

Figure 6.19 – Data source schemas changes propagation process

This task involves the updating of the sets of mapping views $\{M_{Em1}, M_{Em2}, \ldots, M_{Emn}\}$, where M_{Emi} denotes the set of mapping views corresponding to the mediation entity Em_i. When a mapping views evolution process receives a sequence of events from its corresponding data source then it executes the following tasks:

– *Triggering*: this task is executed for each mediation entity and consists in looking into the rule repository and extracting the rules that are triggered by each event.

– *Evaluation*: during this task, the conditions of the rules are evaluated with respect to a given mediation entity. During the condition evaluation, it may be necessary to identify new correspondence assertions between elements in the mediation schema and elements in a data source schema. This task is done by the *Schema Matcher*.

– *Execution*: this task leads to rule execution, which is carried out by firing the evaluated rules and executing the corresponding actions. These actions consist in modifying the mapping views and the set of candidate operations between them. Each mediation entity E_m is associated with two attributes, `MAPSET_STATUS` and `OPSET_STATUS`, defined in section 6.2.2, which are used to determine if the set of mapping views associated with E_m (M_{Em}) and the set of candidates operations to combine these relations (O_{Rm}) were modified during the execution of the rules. These two attributes are set to `False` at the beginning of the evolution process, and they will be set to `True` if a change occurs in the set of mapping views or the set of candidate operations respectively.

– *Notification*: this task notifies the process `P2` about the ending of the mapping views evolution phase.

The mapping views evolution is described by the algorithm presented hereafter. In the following algorithm, consider:

`V:` sequence of events $\{e_1, e_2, \ldots, e_n\}$ notified by a given data source
`T:` set of triggered rules for a given event `e`
`L1:` set of mapping views evolution rules used for propagating data source schemas changes
$Rule_i(E_m)$`.event:` returns the event associated with the $Rule_i$
$Rule_i(E_m)$`.evaluate:` returns the result of the condition evaluation of $Rule_i$ for the mediation entity E_m

```
map_entities_evolution(V)
For each Em in Sm
   For each e in V
     T := ∅
     For each Rulei in L1
         If Rulei(Em).event = e
         Then T := T ∪ Rulei(Em)
     For each Rulei (Em) in T
         If Rulei(Em).evaluate = "TRUE"
         Then execute Rulei(Em)
med_queries_generation(Em)
```

- *Mediation queries generation*

The mediation queries generation consists in generating a mediation query for each mediation entity when the corresponding set of mapping views is modified. Each mediation entity E_m is associated with a process, called mediation queries generation and denoted P2, which is responsible for generating the mediation query used to compute E_m. Note that a mediation entity may become not computable after a data source schema change.

We consider that for a given mediation entity E_m, the process P2 is executed only when all processes P1 have finished the evolution of M_{Em}. This means that the query generation process will start only when all the events notified by all the data sources have been propagated in the corresponding mapping views. The mapping views corresponding to the entity E_m are locked by the processes P1; when these locks are released, the process P2 can lock the mapping views and start its execution. Another possible strategy would consist in executing the process P2 whenever a process P1 signals the end of the evolution of M_{Em}. In this case, the generation process will take place as many often as the number of sequences of events notified by the data sources.

The process of mediation queries generation can be divided into two tasks:

– *Queries generation*: this task consists in generating the set of mediation queries which can be used to compute a given mediation entity E_m. When a process P2 receives a notification from a process P1, then it verifies if either the set of mapping views M_{Em} or the set of operations between them was modified. To do this, the process verifies the values of the attributes MAPSET_STATUS and OPSET_STATUS respectively. If one of them was modified then a new set of mediation queries must be generated for the entity E_m.

  It is important to observe that after this task some entities may become no longer computable, i.e., it is not possible to generate any query to compute the entity using the data available in the local data sources. In this case, it is not necessary to execute the next task (query choice).

– *Query choice*: as the result of the mediation queries generation can be a set of mediation queries, it is necessary to choose one of them to be used to compute the corresponding mediation entity. To do this, the system either interacts with the mediation schema administrator who chooses the best mediation query to compute a given mediation entity, or it uses some heuristics to select a query.

The algorithm described below summarizes the generation of mediation queries associated with a mediation entity. The `search_computation_path`, `generate_query` and `query_choice` primitives were defined in Table 6.2. In the following algorithm, consider:

`P1:` the process corresponding to the mapping view evolution
E_m: an entity in the mediation schema
M_{Em}: the set of mapping views associated with the entity E_m
G_{Em}: the operation graph associated with the entity E_m
`Q:` set of mediation queries

```
med_queries_generation(Em)
While ∃ P1 updating the set M_Em do wait( )
/* the process P2 is executed only when all processes P1
   have finished the evolution of M_Em.*/

If R.OPSET_STATUS = "TRUE" or R.MAPSET_STATUS = "TRUE"
Then search_computation_path(G_Em)
     Q := generate_query(G_Em, Q)
     If Q ≠ ∅
     Then query_choice(Q, q)
          R.OPSET_STATUS = "FALSE"
          R.MAPSET_STATUS = "FALSE"
```

6.6 The users' requirements changes propagation process

This section presents the process of propagating users' requirements changes (Figure 6.20) to the mediation level. In this process, a change event is generated by the *Users' Requirements Manager* and represents a change in the mediation schema. When the *Users' Requirements Manager* receives users' requirements modifications it identifies the change operations to be performed in the mediation schema in order to reproduce them.

Similar to the data source schemas changes propagation process, this process can also be divided into two main tasks: mapping views evolution and mediation queries generation, which will be described in the following.

- *Mapping views evolution*

The mapping views evolution consists in propagating users' requirements changes to the mapping views using the set of ECA rules described in section 6.4. This task is executed by a single mapping view evolution process, which receives mediation schema changes instead of data source schemas changes. We use only one process because, in this case, there is a single source of events, the *Users' Requirements Manager*. Moreover, all changes refer to the mediation schema and affect only the mapping views associated with the mediation entity that is being modified. In this context, an event corresponds to a change operation to be performed in one specific mediation entity.

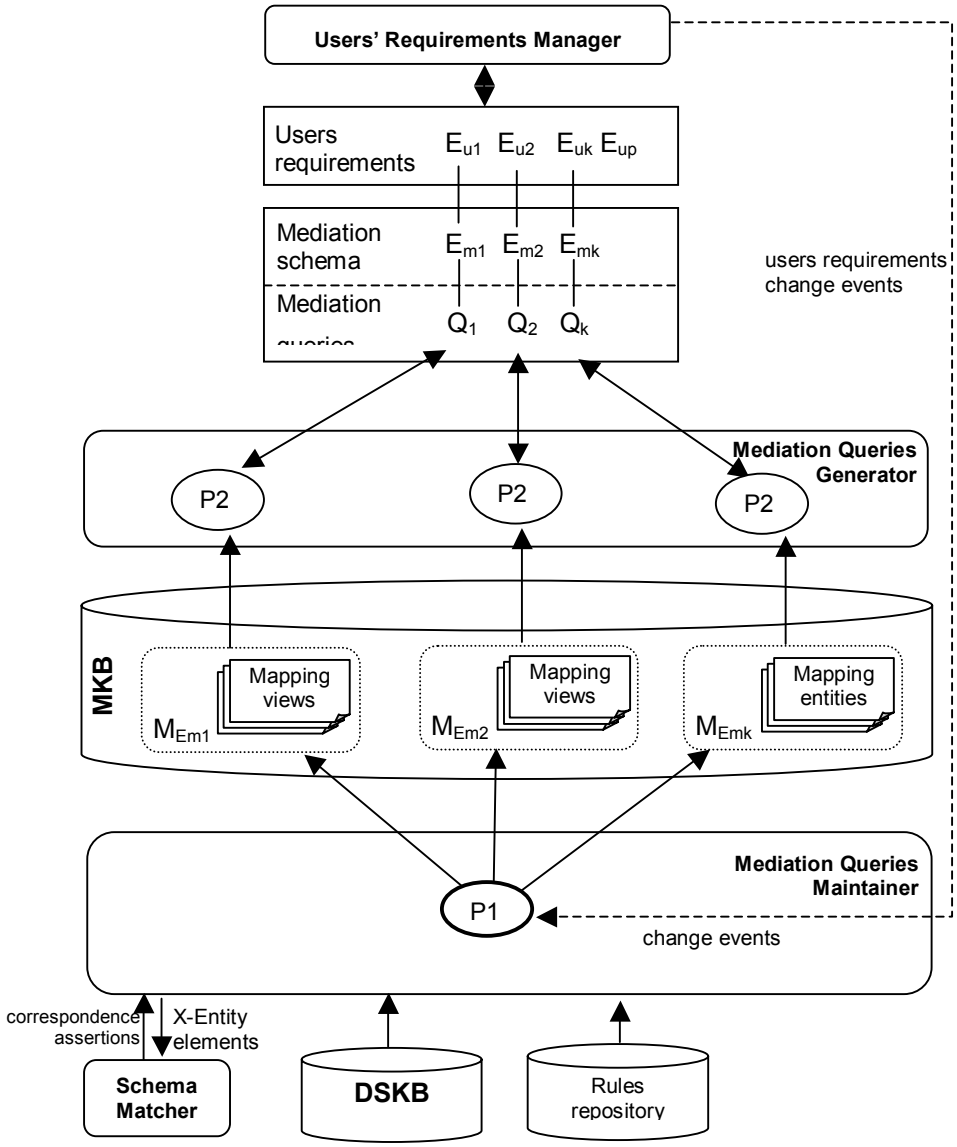

Figure 6.20 – Users' requirements changes propagation process

When the mapping views evolution process receives an event from the *Users' Requirements Manager* then it executes the following tasks:

- *Triggering*: this task consists in looking into the rule repository and extracting the rules that are triggered by the event.
- *Evaluation*: during this task, the conditions of the rules are evaluated with respect to a given mapping view. When the event refers to the addition of a new mediation entity then the whole set of source entities must be considered in the rules evaluation. In this case, some heuristics must be defined to optimize the search for relevant source entities.
- *Execution*: this task leads to rule execution, which is carried out by firing the evaluated rules and executing the corresponding actions. These actions consist in modifying the

mapping views and the set of candidate operations between them. This task is similar to the execution task presented in the earlier section.

– *Notification*: this task notifies the process P2 about the ending of the mapping views evolution phase.

The mapping views evolution is described by the algorithm presented hereafter. In the following, consider:

V: mapping view in M_{Em}
T: set of triggered rules for a given event e
L2: set of mapping views evolution rules used for propagating users' requirements changes

```
map_entities_evolution(e,Em)
If e ≠ add_entity(Em, Sm)
Then For each V in MEm
       T := ∅
       For each Rule in L2
            If Rulei(V).event = e
            Then T := T ∪ Rule(V)
       For each Rulei (V) in T
            If Rulei (V).evaluate = "TRUE"
            Then execute Rulei (V)
Else For each Sj
       For each Eij in Sj.E
         If Rulei (V).evaluate = "TRUE"
         Then execute Rulei(Eij)
med_queries_generation(Em)
```

- *Mediation queries generation*

The mediation queries generation consists in generating a mediation query for a mediation entity when the corresponding set of mapping views is modified. The process of users' requirements change propagation uses the processes P2, described in the earlier section, to generate mediation queries for a given mediation entity E_m after the propagation of a change in the set of mapping views associated with E_m. At the end of the mediation queries generation, if it will be possible to generate a mediation query for E_m then the modifications done in the operation graph G_{Em} and in the mediation schema are confirmed. Otherwise, the modifications performed to reflect the users' requirements change are annulated.

The process of propagating users' requirements changes and the process of propagating data source schemas changes must be executed in parallel in order to maintain the consistency of the mediation queries with respect to the data sources and the users' needs.

6.7 Prototype

We implemented a partial prototype of the proposed data integration system (cf. Chapter 3) in collaboration with the Database Systems Research Group from the Federal University of Pernambuco. To implement the Java classes we used JBuilder 7 from Borland. In this phase, some APIs were coupled to the system, including:

– *JDom*: Java-based solution for accessing, manipulating, and outputting XML data from Java code.

– *DBAccessor*: API used for accessing and extracting schema information from remote databases.

In addition, a database in a server SQL Server 7.0 was created for performing initial tests of the system. Only the functionalities concerned to the mediation queries evolution were implemented, as described below:

- **Lookup.** It periodically communicates with the participating data sources in order to extract a new version of the exported schema. The frequency that determines when the *Lookup* must extract a new version from a given exported schema is defined by the corresponding data source. When a new data source joins the system, it has to provide different configuration data, including its schema change frequency. Currently, the *Lookup* only extracts exported schemas of databases defined in the SQL Server platform. To do this, it uses the API DBAccessor, which allows the access to a remote database and provides support for extracting its schema. XML is the format used by the DBAcessor for communication. In the current version, the *Lookup* does not perform the translation of the exported schema to the XML Schema language.
- **Conceptual Schema Manager**. It receives pre-processed *XML Schemas* and generates X-Entity specifications. It is also capable of comparing different versions of the same conceptual schema in order to identify the schema change operations. After identifying the changes it sends a set of events to the *Mediation Queries Maintainer* and if modifications were performed in the schema it sends the new schema to the DSKB (*Data Sources Knowledge Base*).
- **Mediation Queries Maintainer.** The current version of this module propagates only source schema changes. We are not considering that different processes will be executed in parallel to propagate schema changes from different data sources. We implemented only one mapping views evolution process, which receives events from a data source and propagates them to the various sets of mapping views. To propagate the data source schemas changes,

this process uses a set of ECA rules. These rules were implemented using an abstract class, named `Rule`, and other 27 concrete classes, named `Rule_X`, which have specific properties of each propagation rule. For example, one method `verifyCondition` was implemented for each concrete class `Rule_X` defining the condition to be evaluated. Besides, propagation primitives have been implemented in order to propagate the changes into the operation graphs representing the mediation queries. At the end of the mapping views evolution process, the operation graphs affected by the data source schemas changes will be modified in order to reflect these changes.

- **Mediator Knowledge Base and the Data Sources Knowledge Base.** These knowledge bases were formally specified and implemented. Mediator metadata and data sources metadata are stored in XML documents and two different XML schemas were defined to provide support to their formal specification.

6.8 Concluding remarks

In this chapter, we have presented the process of managing the evolution of mediation queries. Changes to mediation queries may be due to changes in the users' requirements or data source schemas. When data source schemas change or new data sources are added to the system or existing data sources are removed, it is needed to propagate these changes to the mediation queries. As we already discussed, mediation queries are very sensitive to changes over the local data sources.

We also deal with the evolution of users' requirements, which continue to change during the system development as well as when the system is put to use. Users' requirements modifications originate change operations in the mediation schema. If these changes can be reflected in the mediation queries, the modifications on the mediation schema are committed, otherwise the user is informed that his or her new requirements cannot be satisfied.

Data source schemas or users' requirements changes are propagated to the mediation queries through a set of ECA rules, which are triggered according to the different schema changes. The propagation process consists of two main tasks: first, the triggering, evaluation and execution of the rules in order to update the mapping views and the operations among them, and secondly, new mediation queries are generated using the modified operation graphs.

The evolution process can also be seen as a solution for the problem of generating the mediation schema and the mediation queries during the initial phase of the system development. Whenever a new data source joins the system, then the users' requirements are analyzed to identify the mediation entities that are computable. Such entities are inserted into

the mediation schema and their corresponding mediation queries are generated. Moreover, the new data source may have relevant entities for computing existing mediation entities.

The problem of mediation queries evolution is also discussed in other work [Ambite et al. 2001, McBrien et al. 2002, Nica et al. 1999]. The work presented in [Ambite et al. 2001] adopts an approach similar to ours for defining mediation queries. The algorithm to discover integration axioms is incremental, which means that when new sources are added, the system can efficiently update the axioms, but no details on how this could be achieved nor examples are given. In the case of deleting a source the algorithm must start from scratch. They use the LAV approach to define the mappings between the global model and the local sources, while we have adopted the GAV approach.

In [McBrien et al. 2002] an approach is presented to handle both schema integration and schema evolution in heterogeneous database architectures. This approach is different from ours, because instead of defining mediation queries as our approach does, they use primitive transformations to automatically translate queries posed to the global schema to queries over the local schemas. This set of transformations can also be used to systematically adapt the global schema and the global query translation pathways after changes to the source schemas.

Our approach differs from the approach proposed in [Nica et al. 1999], because the modifications are not directly executed in the mediation query definition but in the metadata that describes the mediation query. We consider that a mediation query must be modified as a consequence of every kind of source schema change. In [Nica et al. 1999], a view must evolve only when a source schema change occurs that can make the view definition obsolete; i.e., only the cases of removal of relations or attributes are dealt with. It is important to note that none of the systems, which adopt XML as the common data model, discusses the problem of mediation queries evolution.

One problem to be investigated is the optimization of the synchronization between the update of the mapping views and the generation of mediation queries. One solution consists in generating the mediation queries after the propagation of a single event, but this may degrade the performance of the system. On the other hand, it may be very costly waiting that all events notified by the data sources and the *Users' Requirements Manager* be propagated at the same time. Therefore, it is necessary to solve the problem of determining the sequences of events that must be treated together. As the propagation process is always running in background, other aspects concerning concurrency and failure recovery must also be investigated.

Chapter 7

Conclusion

7.1 Research contributions

Two main approaches are presented in the literature to define the mediation mappings between the data sources and the mediation schema [Halevy 2000, Levy 2000, Ullman 1997]: GAV and LAV. The main differences btween these approaches concern the adaptation of mediation queries in function of source evolution and user queries decomposition. In the GAV approach, mediation queries are very sensitive to changes in the data sources and their adaptation may be a very complex task. On the contrary, in LAV approach the set of mediation queries can be easily adapted after a source schema change. On the other hand, in GAV approach user queries decomposition is a very simple task while in LAV approach this process is very complicated and time-consuming. In this context, we have proposed a solution to the problem of managing the evolution of mediation queries for data integration systems which adopt the GAV approach.

The basis for our solution is the adopted process of mediation queries generation, which provides a formalism to represent mediation queries. This formalism is based on the concept of operation graphs, which describes the relevant sources to compute a given mediation entity and the possible ways of combining them. This high-level representation facilitates the identification of the mediation entities that are affected by a data source schema change and that, consequently, should be rewritten. Moreover, it becomes easier to propagate data source changes into mediation queries. Using this approach, the problem of mediation queries evolution becomes the problem of maintaining the operation graphs that describe the mediation queries.

One advantage of our approach is that the mediation level (mediation schema and mediation queries) can be developed incrementally based on the evolution of the local data sources and the evolution of the users' requirements, which means that if the data source schemas or the users' requirements change frequently the mediation level can be updated to reflect these changes. The proposed approach allows the mediation level to evolve and modifications can be handled easier by increasing the system flexibility and scalability. The main contributions of our work are summarized below:

- We have proposed an architecture for a mediator-based data integration system, which also deals with the problems concerning generation of mediation queries and maintenance of the data integration system. Another distinguishing feature of the system is that the queries are executed using three kind of data: i) virtual data, which are obtained on demand and accessed directly from the data sources, ii) materialized data, which are obtained from the data warehouse over selectively materialized data and iii) cached queries results, which are answered queries and previously stored in a local cache. Some other key issues of our system are: the use of XML as the common model for data exchange and integration, and the use of XML Schema as the language for representing the mediation schema and the exported schemas. XML has been adopted by the most recent data integration systems proposals as the common data model for data integration due to its flexibility to represent both structured and semi-structured information, which are very common in the data integration context.

- To provide a high-level abstraction for information described in an *XML Schema* we have defined a conceptual data model, called X-Entity model. The X- Entity model is not a new formalism for conceptual modeling; rather it is an extension of the ER model. We also described the process of converting an *XML Schema* to an X-Entity schema. Other advantage of the X-Entity model is that it is not a model for representing a specific type of XML schema (ex: DTD or *XML Schema*). The X-Entity model may be used during the conceptual design of an XML schema and later the X-Entity schema can be mapped to a DTD or to an *XML Schema*, for example.

- We extended the approach proposed by Kedad & Bouzeghoub [Kedad et al. 1999], which specifies how to generate computing expressions for relational views, by: i) redefining the process of identifying the relevant source entities, ii) specifying new operators to be applied between the mapping views and iii) redefining the process of generating computing expressions. In our approach, the mediation schema is represented by an X-Entity schema and the process of mediation queries generation consists in discovering a computing

expression for each entity in the mediation schema. Based on the mediation queries, generated off-line, the system computes at run-time the most appropriate rewriting for answering a user query by simply executing the necessary mediation queries and combining their results.

- We have defined a set of local source change operations representing the possible evolutions at the source level or at the user level and a set of propagation primitives reflecting the changes at the mediation level. To specify the propagation of schema changes, we have proposed a set of ECA rules. We have also discussed the propagation process and described its two main tasks, the mapping entities evolution and the mediation queries generation.

7.2 Future work

This work has opened a large spectrum for new problems to be solved, which are listed below:

- *Mediation schema definition* : in this work we do not formalize the process of mediation schema definition. We assume that the mediation schema is defined comparing the users' requirements and the data source schemas, in such a way that the mediation schema consists of the users' requirements which can be answered from the set of available data sources. Also, we have to specify the propagation process of data source schemas changes into the users' requirements schema to identify the users' needs that become computable after these changes.
- *X-Entity model extension*: some characteristics of XML Schema were not considered in the process of converting an XML Schema to its corresponding X-Entity schema, including: hierarchy, cardinality of groups and attributes ID and IDREF. To consider such features, the X-Entity model must be extended with new concepts and the conversion process must be extended with new rules.
- *X-Entity schemas matching*: we do not adopt any specific process for matching X-Entity schemas. A deeply investigation of the existing methodologies for schema matching must be performed in order to identify if one of them is suitable for the matching of X-Entity schemas.
- *User queries computation*: we presented how to use mediation queries to compute mediation entities in function of the distributed data. However, we did not present a detailed description of how to use mediation entities to compute user queries. This is a very interesting work, which consists of identifying the relevant mediation entities to compute a

given user query and identifying how to combine them in order to obtain the requested answer.

- *Definition of heuristics to select mediation queries*: as proposed in [Kedad et al 1999], some heuristics must be defined in order to optimize the process of choosing the best query to compute a given mediation entity.
- *Mediation queries quality evaluation*: another perspective is to take into account some quality criteria to evaluate the impact of the changes in the local data sources on the quality of the mediation schema. Quality factors could be applied to analyze the evolution of mediation queries on data integration systems. This is an important issue because it directly impacts the quality of the data and the services offered by the system.
- *Optimization of the propagation process*: as the number of dynamic data source increases the process of propagating data source schemas changes becomes more complex. In this context, a better synchronization of the mapping entities evolution and the mediation queries generation is necessary. The propagation process must be optimized in order to minimize the cost of mediation queries evolution. We also intend to analyze how to keep a history of the exported schema updates in order to minimize the impact of new updates in the integration system.
- *Minimal rule set*: data source schemas or users' requirements changes are propagated to the mediation queries through a set of ECA rules, which are triggered according to the different schema changes. It is then natural to ask whether there is a "minimal" or smallest set of rules necessary to propagate the different schema changes. In this work we do not answer this question. An interesting future work consists in defining one or more criteria for redundancy of a rule in such a way that this minimal rule set achieves the same update propagation.
- *Stepwise consistency*: the propagation process consists of two main tasks: first, the triggering, evaluation and execution of the rules in order to update the mapping entities and the operations among them, and secondly, new mediation queries are generated using the modified operation graphs. In this work, we assume that the output set of mediation queries is consistent, i.e, we do not investigate if the new mediation queries set satisfies the constraints specified in the user requirements schema as well as any other constraints that should hold in the mediation schema. Another important future work is to prove consistency of a collection of mediation queries obtained after the propagation process.

Bibliography

[Abiteboul et al. 1997] ABITEBOUL, S., J.MCHUGH, D., WIDOM, J., WIENER, J., The lorel query language for semistructured data, **International Journal on Digital Libraries**, v. 1, n. 1, p. 68-88, 1997.

[Abiteboul et al. 1998] ABITEBOUL, S., MCHUGH, J., RYS, M., VASSALOS V., WEINER, J. Incremental maintenance for materialized views over semistructured data, In: 24th International Conference on Very Large Data Bases, New York City, New York,1998, **Proceedings...,** p. 38-49.

[Abiteboul et al. 2000] ABITEBOUL, S., BUNEMAN, P., SUCIU, D., **Data on the Web**, 1st. Ed. Morgan Kaufmann Publishers, 2000, 258p.

[Ambite et al. 1998] AMBITE, J., ASHISH, N., BARISH, G., KNOBLOCK, A. C., MINTON, S., MODI, P., MUSLEA, I., PHILPOT, A., TEJADA, S. Ariadne: a system for constructing mediators for internet sources, In: ACM SIGMOD Conference on Management of Data, Seattle, Washington, 1998, **Proceedings...**, p. 561-563.

[Ambite et al. 2001] AMBITE, J., KNOBLOCK, C., MUSLEA, I., PHILPOT, A. Compiling Source Description for Efficient and Flexible Information Integration, **Journal of Intelligent Information Systems**, v. 16, n. 2, p. 149-187, 2001.

[Arens et al. 1993] ARENS, V., CHEE, C. Y., HSU, C-N., KNOBLOCK, C. A. Retrieving and integrating data from multiple information sources, **International Journal on Intelligent and Cooperative Information Systems**, v. 2, n. 2, p. 127-158, 1993.

[Banerjee et al. 1987] BANERJEE, J., KIM, W., KIM, H. J., KORTH, H. F. Semantics and Implementation of schema evolution in object-oriented databases, In: SIGMOD, San Francisco, California, 1987, **Proceedings…**, p. 311-322.

[Baru et al. 1999] BARU, C., GUPTA, A., LUDASCHER, B., MARCIANO, R., PAPAKOSTATINOU, Y., VELIKHOV, P., CHU, V. Xml-based information mediation with mix, In: ACM SIGMOD Conference on Management of Data, Philadelphia, Pennsylvania, 1999, **Proceedings...,** p. 597-599.

[Baru 1999] BARU, C. Xviews: XML Views of Relational Schemas, In: International Workshop on Internet Data Management, Florence, Italy, 1999, **Proceedings…**, p. 700-705.

[Batista et al. 2003] BATISTA, M. C. M., LÓSCIO, B.F., SALGADO, A. C. Optimizing access in a data integration system with caching and materialized data, In: 5th International Conference on Enterprise Information Systems – ICEIS, Angers, France, 2003, **Proceedings…**, p. 529-532.

[Batista 2003] BATISTA, M. C. M. Otimização de Acesso em um Sistema de Integração de Dados através do uso de Caching e Materialização de Dados, Master Thesis, Federal University of Pernambuco, 2003.

[Beech et al. 1999] BEECH, D., MALHOTRA, A., RYS, M. A formal data model and algebra for XML, Communication to the W3C, 1999.

[Beneventano et al. 2001] BENEVENTANO, D., BERGAMASCHI, S., MANDREOLI, F. Extensional Knowledge for Semantic Query Optimization in a Mediator Based System, In: International Workshop on Foundations of Models for Information Integration, 2001, **Proceedings...**

[Bergamaschi et al. 1998] BERGAMASCHI, S., CASTANO, S., DE CAPITANI DI VIMERCATI, S., MONTANARI, S., VINCINI, M. A semantic approach to information integration: the momis project, In: Sesto Convegno della Associazione Italiana per l'Intelligenza Artificiale, **Proceedings...**, 1998.

[Bird et al. 2000] BIRD, L., GOODCHILD, A., HALPIN, T. Object Role Modeling and XML-Schema, In: International Conference on Conceptual Modeling (ER), Salt Lake City, Utah, 2000, **Proceedings…**, p. 309-322.

[Biron et al. 2001] BIRON, P. V., MALHOTRA, A. XML Schema Part 2: Datatypes, World Wide Web Consortium. Available at: http://www.w3.org/TR/xmlschema-2, May 2001.

[Bohannon et al. 2002] BOHANNON, P., FREIRE, J., ROY, P., SIMEON, J., From XML Schema to Relations: A Cost-Based Approach to XML Storage, In: 18th International Conference on Data EngineeringICDE, San Jose, CA, 2002, **Proceedings…**, p. 64-.

[Bonifati et al. 2000] BONIFATI, A., CERI, S. Comparative Analysis of Five XML Query Languages, **Journal SIGMOD Record**, v .29, n.1, p. 68-79, 2000.

[Booch et al. 1999] BOOCH, G., CHRISTERSON, M. , FUCHS, M., KOISTINEN, J. UML for XML Schema Mapping Specification. Rational Website, 1999. Available at: http://www.rational.com/media/ uml/resources/media/uml_ xmlschema33.pdf.

[Bouzeghoub et al. 2002] BOUZEGHOUB, M., LÓSCIO, B. F., KEDAD, Z., SOUKANE, A. Heterogeneous Data Source Integration and Evolution, (Extended Abstract), In: 13th International Conference DEXA, 2002, **Proceedings…**,p. 751-757.

[Bray et al. 2000] BRAY, T., PAOLI, J., SPERBERG-MCQUEEN, C. M. Extensible Markup Language (XML) 1.0, World Wide Web Consortium. Available at: http://www.w3.org/TR/REC-xml, October 2000.

[Braganholo 2002] BRAGANHOLO, V. Updating relational databases through xml views, Thesis proposal PPGC-UFRGS, Porto Alegre, 2002.

[Buneman 1997] BUNEMAN, P. Semi-structured data, In: 16th ACM Symposium on Principles of Database Systems, Tucson, Arizona, 1997, **Proceedings...**, p. 117-121.

[Cali et al. 2003] CALI, A., CALVANESE, D., DE GIACOMO, G., LENZERINI, M., NAGGAR, P., VERNACOTOLA, F. IBIS: Semantic Data Integration at Work, In: Advanced Information Systems Engineering, 15th International Conference, CAiSE 2003, Klagenfurt, Austria, 2003, **Proceedings…**, p. 79-94.

[Carey et al. 2000] CAREY, M., FLORESCU, D., IVES, Z., LU, Y., SHANMUGASUNDARAM, J., SHEKITA, E., SUBRAMANIAN, S. XPERANTO: Publishing object-relational data as XML, In: International Workshop on Web and Databases (WebDB), Dallas, Texas, 2000, **Proceedings…**, p. 105-110.

[Cattell et al. 2000] CATTELL, R.G.G., BARRY, D.K.. The Object Database Standard: ODMG

3.0., Morgan Kaufmann, 2000.

[Ceri et al. 1999] CERI, S., COMAI, S., DAMIANI, E., FRATERNALI, P., PARABOSCHI S., TANCA, L. Xml-gl: a graphical language for querying and restructuring www data, In: 8th

International World Wide Web Conference, WWW8, Toronto, Canada, 1999, **Proceedings...**, p. 151-165.

[Chamberlin et al. 2001] CHAMBERLIN, D., FLORESCU, D., ROBIE, J., SIMÉON, J., STEFANESCU, M. XQuery: A Query Language for XML, World Wide Web Consortium. Available at: http://www.w3.org/TR/2001/WD-xquery-20010607/, 2001.

[Chawathe et al. 1994] CHAWATHE, S., GARCIA MOLINA, H., HAMMER, J. The tsimmis project: integration of heterogeneous information sources, In: 10th Meeting of the Information Processing Society of Japan (IPSJ), 1994, **Proceedings...**, p. 7-18.

[Chen 1976] CHEN, P.P. The Entity-Relationship Model: Toward a unified view of data, ACM Transactions on Database Systems, v. 1, n. 1, p. 1-36, 1976.

[Chinwala et al. 2001] CHINWALA, M., MILLER, J. A. Algebraic Languages for XML Databases, Technical Report, University of Georgia, 2001.

[Clark 1999a] CLARK, J. Xml Path Language (XPATH), World Wide Web Consortium, Available at: http://www.w3.org/TR/xpath, November 1999.

[Clark 1999b] CLARK, J. XSL Transformations (XSLT specification), World Wide Web Consortium. Available at: http://www.w3.org/TR/WD-xslt, November 1999.

[Clark 2000] CLARK, J. TREX - Tree Regular Expressions for XML, Internet Document. Available at: http://www.thaiopensource.com/trex/tutorial.html, January 2000.

[Claypool et al. 1998] CLAYPOOL, K. T., JIN, J., RUNDENSTEINER, E. A., SERF: Schema Evalution through an Extensible, Re-usable and Flexible Framework, In: ACM CIKM International Conference on Information and Knowledge Management, Bethesda, Maryland, 1998, **Proceedings…,** p. 314-321.

[Cluet et al. 1998] CLUET, S., DELOBEL, C., SIMÉON, J., SMAGA., K. Your mediators need data conversion!, In: ACM SIGMOD Conference on Management of Data, Seattle, Washington, 1998, **Proceedings...**, p. 177-188.

[Christophides et al. 2000] CHRISTOPHIDES V., CLUET, S., SIMÉON, J. On Wrapping Query Languages and Efficient XML Integration, In: SIGMOD Conference, Dallas, Texas, 2000, **Proceedings...**, p. 141-152.

[Dar et al. 1996] DAR, S., FRANKLIN, M.J., JÓNSSON, B.T., SRIVASTAVA, D., TAN, M. Semantic Data Caching and Replacement, In: 22nd VLDB Conference, Mumbai (Bombay), Índia, 1996, **Proceedings…**, p. 330-341.

[Davidson et al. 1999] DAVIDSON, A., FUCHS, M., HEDIN, M., JAIN, M., KOISTINEN, J., LLOYD, C., MALONEY, M., SCHWARZHOF, K. Schema for Object-Oriented XML 2.0, World Wide Web Consortium. Available at: http://www.w3.org/TR/NOTE-SOX, July 1999.

[Deutsch et al. 1999] DEUTSCH, A., FERNANDEZ, M., FLORESCU, D., LEVY, A. AND SUCIU, D. A query language for xml, In: International World Wide Web Conference, Toronto, Canadá, 1999, **Proceedings...**,p. 11-16.

[Draper et al. 2001] DRAPER, D., HALEVY, A. Y., WELD, D. S., The Nimble XML Data Integration System, In: 17th International Conference on Data Engineering, , Heidelberg, Germany, 2001, **Proceedings...**, p. 155-160.

[Draper et al. 2002] DRAPER, D., FANKHAUSER, P., FERNANDEZ, M., SIMÉON, J., MALHOTRA, A., ROSE, K., RYS, M., WADLER, P. XQuery 1.0 and XPath 2.0 Formal Semantics. World Wide Web Consortium. Available at: www.w3.org/TR/query-semantics, November 2002.

[Elmagarmid et al. 1999] ELMAGARMID, A., RUSINKEIWICZ, SHETH, A. **Management of Heterogeneous and Autonomous Database Systems**, 1st. Ed. Morgan Kaufmann Publishers, 1999.

[Fallside 2001] FALLSIDE, D., C. XML Schema Part 0: Primer, World Wide Web Consortium.

Available at: http://www.w3.org/TR/xmlschema-0/, May 2001.

[Fankhauser et al. 2001] FANKHAUSER, P., FERNANDEZ, M., SIMÉON, J., MALHOTRA, A., RYS, M., WADLER, P. The XML Query Algebra, World Wide Web Consortium. Available at: http://www.w3.org/TR/2001/WD-query-algebra-20010215/, February 2001.

[Fernandez et al. 2000] FERNANDEZ, M., TAN, W.-C. , SUCIU, D. SilkRoute: Trading between relations and XML, In: 9th International World Wide Web Conference, Amsterdam, The Netherlands, 2000, **Proceedings…**, p. 723-745.

[Fernandez et al. 2001] FERNANDEZ, M., SIMEON, J., WADLER, P. A semi-monad for semi-structured data, In: International Conference on Database Theory, , 2001, **Proceedings…**, p. 263-300.

[Fernandez et al. 2001] FERNANDEZ, M., MARSH, J. XQuery 1.0 and XPath 2.0 Data Model World Wide Web Consortium. Available at: www.w3.org/TR/query-datamodel/, November 2002.

[Ferrandina et al. 1994a] FERRANDINA, F., MEYER, T., ZICARI, R. Correctness of lazy database updates for an object database system, In: 6th Int'l Workshop on Persistent Object Systems, 1994, **Proceedings…**, p. 284-301.

[Florescu et al. 1998] FLORESCU, D., LEVY A., MENDELZON, A. Database techniques for the world wide web: a survey, **ACM SIGMOD Record**, v. 27, n. 3, p. 59-74, 1998.

[Florescu et al. 1999] FLORESCU, D, KOSSMANN, D. A performance evaluation of alternative mappings schemes for storing XML in a relational database. Technical Report 3680, INRIA, 1999.

[Galanis et al. 2001] GALANIS, L., VIGLAS, E., DEWITT, D. J., NAUGHTON, J. F., MAIER, D. Following the paths of XML data: an algebraic framework for XML query evaluation, Technical Report, University of Wisconsin, Madison, 2001.

[Gardarin et al. 2002]GARDARIN, G., MENSCH, A., TOMASIC, A., An Introduction to the e-XML Data Integration Suite, In: 8th International Conference on Extending Database Technology, Prague, Czech Republic, 2002, **Proceedings...**, p. 297-306.

[Genesereth et al. 1997] GENESERETH, M., KELLER, A., DUSHKA, O. Infomaster: an information integration system, In: ACM SIGMOD, Tucson, Arizona, 1997, **Proceedings...**, p. 539 -542.

[Goasdoué et al. 2000] GOASDOUÉ, F., LATTES, V., ROUSSET, M-C., The Use of CARIN Language and Algorithms for Information Integration: The PICSEL Project, **International Journal of Cooperative Information Systems** (IJCIS), 2000.

[Gupta et al. 1995a] GUPTA, A., MUMICK, I. S. Maintenance of materialized views: problems, techniques, and applications, **IEEE Data Engineering Bulletin**, v. 18, n. 2, p.3-18, 1995.

[Gupta et al. 1997] GUPTA, A., BLAKELEY, J.A. Using partial information to update materialized views, **Information Systems**, v. 20, n. 8, p. 641-662, 1995.

[Halevy 2000] HALEVY, Y., Theory of answering queries using views, **SIGMOD Record**, v. 29, n. 4, p. 40-47, 2000.

[Halphin 1998] HALPHIN, T. Object-Role Modeling (ORM/NIAM), Handbook on Architectures of Information Systems, Springer-Verlag, 1998.

[Hammer et al. 1998] HAMMER, J., GARCIA-MOLINA, H., NESTOROV, S., YERNENI, R., BREUNIG, M. M. VASSALOS, V. Template-based wrappers in the tsimmis system, In: ACM SIGMOD Conference on Management of Data, Tucson, Arizona, 1998, **Proceedings...**, p. 532-535.

[Harinarayan et al. 1996] HARYNARAYAN, V., RAJARAMAN, A., ULLMAN, J.D. Implementing data cubes efficiently, In: ACM SIGMOD Conference, Montreal, Quebec, Canada 1996, **Proceedings…**, p. 205-216.

[Hull 1997] HULL, R. Managing semantic heterogeneity in databases: a theoretical perspective, In: ACM Symposium on Principles of Database Systems, Tucson, Arizona, 1997, **Proceedings...**, p. 51-61.

[Huyn 1997] HUYN, N. Multiple view-self maintenance in data warehousing environments, In: 23rd Very Large Data Bases Conference, Athens, Greece, 1997, **Proceedings...**, p. 26-35.

[Ives et al. 1999] IVES, Z. G., FLORESCU, D., FRIEDMAN, M., LEVY A., WELD, D. S., An adaptive query execution system for data integration, **IEEE Data Engineering Bulletin**, v. 23, n. 2, p. 19-26, 2000.

[Jagadish et al. 2001] JAGADISH, H. V., LAKSHMANAN, L. V. S., SRIVASTAVA, D., THOMPSON, K. TAX: A Tree Algebra for XML, In: Database Programming Languages, 8th International Workshop , DBPL'01, Frascati, Italy, 2001, **Proceedings…**, p. 149-164.

[Jagadish et al. 2002] JAGADISH, H., AL-KHALIFA, S., LAKSHMANAN, L., NIERMAN, A., PAPARIZOS, S., PATEL, J., SRIVASTAVA, D., WU, Y. Timber: A native XML database, Technical Report, University of Michigan, 2002.

[Jellife et al. 2000] JELLIFE, R. Schematron, Internet Document. Available at: http://www.ascc.net/ xml/resource/ schematron/, 2000.

[Kedad et al. 1999] KEDAD, Z., BOUZEGHOUB, M., Discovering View Expressions from a Multi-Source Information System, In: Fourth IFCIS International Conference on Cooperative Information Systems (CoopIS), Edinburgh, Scotland, 1999, **Proceedings...**, p. 57-68.

[Kim 1995] KIM, W., **Modern Database Systems**, Addison-Wesley Pub. Co., 1995.

[Klarlund et al. 2000] KLARLUND, N., MOLLER, A., SCHWATZBACH, M. I. DSD: a schema language for xml, In: 3rd ACM Workshop on Formal Methods in Software Practice, 2000, **Proceedings....**

[Kirk 1995] KIRK, T., LEVY, A.Y., SAGIV ,Y., SRIVASTAVA, D., The Information Manifold, In: Spring Symposium on Information Gathering from Heterogeneous, Distributed Environments, 1995, **Proceedings...**, p. 85-91.

[Lautemann 1997] LAUTEMANN, S. E. Schema Versions in Object Oriented Database Systems, In: International Conference on Database Systems for Advanced Applications (DASFAA), Melbourne, Australia , 1997, **Proceedings...,**p. 323-332.

[Lee et al. 2000] LEE D., CHU, W.W. Comparative analysis of six xml schema languages, **SIGMOD Record**, v. 29, n. 3, p. 76-87, 2000.

[Lerner 1996] LERNER, B. S. A model for compound type changes encountered in schema evolution, Technical Report, University of Massachusetts, Amherst, Computer Science Department, 1996.

[Levy et al. 1995] LEVY, A. Y., MENDELZON, A. O., SAGIV, Y., SRIVASTAVA, D. Anwering queries using views, In: ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS), San Jose, CA, 1995, **Proceedings...**, p. 95-104.

[Levy et al. 1996] LEVY, A. Y., RAJARAMAN, A., ULLMAN, J. D. Answering Queries Using Limited External Processors, In: Symposium on Principles of Database Systems – PODS, Montreal, Canada , 1996, **Proceedings…**, p. 227-237.

[Levy 1999] LEVY, A. Y. Combining artificial intelligence and databases for data integration, **Artificial Intelligence Today**, p. 249-268, 1999.

[Levy 2000] LEVY, A., Y., Logic-based techniques in data integration, In J. Minker, editor Logic based Artificial Intelligence. Kluwer Publishers, 2000.

[Lóscio et al. 2001] LÓSCIO, B. F., SALGADO, A. C., VIDAL, V. M. P. Using Agents for Generation and Maintenance of Mediators in a Data Integration System on the Web, In: XVI Simpósio Brasileiro de Banco de Dados, Rio de Janeiro, Brazil, 2001, **Proceedings...**, p. 172-186.

[Lóscio et al. 2002a] LÓSCIO, B. F., SALGADO, A. C., VIDAL, V.M.V. Quality-driven evolution of mediation systems, In: I Workshop de Teses e Dissertações de Banco de Dados, Rio Grande do Sul, Brazil, 2002, **Proceedings…,** p. 62-66.

[Lóscio et al. 2002b] LÓSCIO, B.F., BOUZEGHOUB, M., KEDAD, Z., SALGADO, A. C. Managing the Evolution of Mediation Queries, Technical Report, Laboratoire PRiSM, Université de Versailles, 2002.

[Lóscio et al. 2003] LÓSCIO, B. F., SALGADO, A. C., GALVÃO, L. R. Conceptual Modeling of XML Schemas, submitted to International Conference on Conceptual Modeling (ER) 2003.

[Madhavan et al. 2001] MADHAVAN, J., BERNSTEIN, P. A., RAHM, E. Generic Schema Matching with Cupid, In: 27th International Conference on Very Large Data Bases, Roma, Italy, 2001, **Proceedings…**,p. 49-58.

[Makoto 2000]MAKOTO, M. RELAX (REgular LAnguage description for XML), Internet Document. Available at: www.xml.gr.jp/**relax**/, April 2000.

[Mani et al. 2001] MANI, M., LEE, D., MUNTZ, R. Semantic Data Modeling using XML Schemas, In: 20th International Conference on Conceptual Modeling (ER), Roma, Italy, 2001, **Proceedings…**, p. 149-163.

[Marche 1993] MARCHE, S. Measuring the Stability of Data Models, **European journal of Information Systems**, v. 2, n. 1, p. 37-47, 1993.

[McBrien et al. 2002] MCBRIEN, P., POULOVASSILIS, A. Schema Evolution in Heterogeneous Database Architectures, A Schema Transformation Approach, In: CAiSE'02, Toronto, Canadá, 2002, **Proceedings…**, p. 484-499.

[McHugh et al. 1999] MCHUGH, J., WIDOM, J. Query Optimization for XML. In: Twenty-Fifth International Conference on Very Large Databases, Edinburgh, Scotland, 1999, **Proceedings…**, p. 315-326 .

[Mello et al. 2001] MELLO, R. S., HEUSER, C. A., A Rule-Based Conversion of a DTD to a Conceptual Schema, In: 20th International Conference on Conceptual Modeling (ER), Roma, Italy, 2001, **Proceedings…**, p. 133-148.

[Mello et al. 2002] MELLO, R., CASTANO, S., HEUSER, C. A Method for the Unification of XML Data, **Information & Software Technology**, v. 44, n. 4, p. 241-249.

[Mena et al. 1996] MENA, E., KASHYAP, V., SHETH, A., ILLARRAMENDI, A., OBSERVER: an approach for query processing in global information systems based on interoperation across pre-existing ontologies, In: 4th Conference on Cooperative Systems, Brussels, Belgium, 1996, **Proceedings...**, p.14-25.

[Miller et al. 1994] MILLER, R., IOANNIDIS, Y.E., RAMAKRISHNAN, R. Schema equivalence in heterogeneous systems: bridging theory and practice, **Information Systems**, v. 19, n.1, p. 3-31.

[Milo et al. 1998] MILO, T., ZOHAR, S. Using Schema Matching to Simplify Heterogeneous Data Translation, In: 24rd International Conference on Very Large Data Bases, New York City, New York , 1998, **Proceedings…**, p. 122-133.

[Naumann et al. 1999] NAUMANN, F., LESER, U., Quality-driven integration of heterogeneous information systems, In: 25th Very Large Databases Conference (VLDB). Edinburh, Scotland, 1999, **Proceedings...**, p. 447-458.

[Nica et al. 1999] NICA, A., RUNDENSTEINER, E. A. View maintenance after view synchronization, In: International Database Engineering and Application Symposium (IDEAS'99), Montreal, Canada , 1999, **Proceedings...**, p. 215-213.

[Nica 1999] NICA, A. View evolution Support for information integration systems over dynamic distributed information spaces, Ph.d Dissertation Electrical Engineering and Computer Science Department, University of Michigan, Ann Arbor, 1999.

[Papakonstantinou et al. 1996] PAPAKONSTANTINOU, Y., ABITEBOUL, S., GARCIA-MOLINA, H. Object Fusion in Mediator Systems, In: Twenty-second International Conference on Very Large Databases (VLDB), Mumbai (Bombay), India., 1996, **Proceedings…**, p. 413-424.

[Psaila 2000] PSAILA, G., ERX: A Conceptual Model for XML Documents, In: ACM Symposium on Applied Computing (SAC), Villa Olmo, Italy, 2000, **Proceedings…**, p. 898-903.

[Passi et al. 2002] PASSI, K., MADRIA, S., S., BIPIN, MOHANIA, BHOWMICK, M., S., A Model for XML Schema Integration, In: 3rd International Conference on E-commerce and Web Technology, France, 2002, **Proceedings…**, p. 193- 202.

[Pottinger et al. 2000] POTTINGER, R., LEVY, A. A Scalable Algorithm for Answering Queries Using Views, In: 26th International Conference on Very Large Data Bases (VLDB), Cairo, Egypt , 2000, **Proceedings…**, p. 484-495.

[Ra et al. 1997] RA, Y. G., RUNDENSTEINER, E. A., A Transparent Schema Evolution System Based on Object-Oriented View Technology, **IEEE Transactions on Knowledge and Data Engineering**, v. 9, n. 4, p. 600-624, 1997.

[Rahm et al. 2001] RAHM, E., BERNSTEIN, P. A. A survey of approaches to automatic schema matching, **Very Large Data Bases Journal**, v. 10, n. 4, p. 334-350, 2001.

[Robie 1998a] ROBIE, J. The design of XQL, World Wide Web Consortium. Available at: http://www.w3.org/ Style/XSL/Group/1998/09/XQL-design.html, 1998.

[Robie et al. 1998b] ROBIE, J., LAPP, J. SCHACH, D. Xml query language (xql), In: Query Languages workshop, Cambridge, Mass, 1998, **Proceedings...**.

[Rundensteiner et al. 1997] RUNDENSTEINER, E. A., LEE, A. J., NICA, A. On preserving Views In: evolving environments, In: 4th KRDB Workshop, Athens, Greece, 1997, **Proceedings...**, p. 13.1-13.11.

[Rundensteiner et al. 1998] RUNDENSTEINER, E.A., LEE, A. RA, Y.G. Capacity augmenting schema changes on object oriented databases: Towards increased interoperability, In: ObjectOriented Information Systems, 1998.

[Rundensteiner et al. 2000] RUNDENSTEINER, E., KOELLER, A. AND ZHANG, X. Maintanining Data Warehouses over Changing Information Sources, **Communications of the ACM**, v. 43, n.1, p. 57-62, 2000.

[Schach et al. 1998] SCHACH, D., LAPP, J., ROBIE, J. Querying and transforming xml, In: Query Languages workshop, Cambridge, Mass, 1998, **Proceedings....**

[Shanmugasundaram et al. 2001] SHANMUGASUNDARAM, J., SHEKITA, E., BARR R., CAREY, M., LINDSAY, B., PIRAHESH , H., REINWALD, B. Efficiently publishing relational data as XML documents, **Very Large Data Bases Journal,** v. 10, n. 2-3, p. 133-154, 2001.

[Sheth et al. 1990] SHETH, A. P. AND LARSON, J. A federated database systems for managing distributed, heterogeneous, and autonomous databases, **Computing Surveys**, vol. 22, no. 3, p.183-236, 1990.

[Sjoberg 1993] SJOBERG, D. Quantifying Schema Evolution. **Information and Software Technology**, vol. 35, no. 1, p. 35-54, Jan. 1993.

[Spaccapietra et al. 1994] SPACCAPIETRA, S. AND PARENT, C. View integration: a step forward in solving structural conflicts, **IEEE Transactions on Knowledge and Data Engineering**, vol. 6, no. 2, 1994.

[Srivastava et al. 1996] SRIVASTAVA, D., DAR, S., JAGADISH, H. V., AND LEVY, A. Y. Answering queries with aggregation using views, In: 22nd International Conference on Very Large Databases, Mumbai (Bombay), India, 1996, **Proceedings...**, p. 318-329, 1996.

[Theodoratos et al. 1997] THEODORATOS, D., SELLIS, T. Data warehouse configuration, In: 23rd VLDB Conference, Athens, Greece, 1997, **Proceedings…**, p. 126-135.

[Tomasic 1998] TOMASIC, A., RASCHID, L., AND VALDURIEZ, P., Scaling access to distributed heterogeneous data sources with Disco, **IEEE Transactions on Knowledge and Data Engineering**, 1998.

[Thompson et al. 2000] THOMPSON, H. S., BEECH, D., MALONEY, M. AND MENDELSOHN, N. XML Schema Part 1: Structures, World Wide Web Consortium, http://www.w3.org/TR/xmlschema-1, 2000.

[Tompa et al. 1988] TOMPA, F.W. AND BLAKELEY, J. A. Maintaining materialized views without accessing base data**, Informations Systems**, vol. 13, no. 4, p. 393-406, 1988.

[Ullman 1997] ULLMAN, J. D., Information integration using logical views, In: ICDT'97, Delphi, Greece, 1997, **Proceedings…**, p. 19-40.

[Vidal et al. 2001] VIDAL, V. M. P., LÓSCIO, B. F. AND SALGADO, A. C. Using correspondence assertions for specifying the semantics of xml-based mediators, In: WIIW 2001 - International Workshop on Information Integration on the Web - Technologies and Applications, Rio de Janeiro, Brasil, 2001, **Proceedings...**, p. 3-10.

[Vittori et al. 2001] VITTORI, C., DORNELES, C., HEUSER, C. Creating xml documents from relational data sources, In: Second International Conference Electronic Commerce and Web Technologies EC-Web, Munich, Germany ,2001, **Proceedings…**, p. 60-70.

[Zhang et al. 2002a] ZHANG, X., MULCHANDANI, M., CHRIST, S., MURPHY, B., RUNDENSTEINER, E. A. Rainbow: Mapping-Driven XQuery Processing System, In: Demo Session of SIGMOD'02, Madison, Wisconsin, 2002, **Proceedings…**, p. 614.

[Zhang et al. 2002b] ZHANG, X., PIELECH, B. AND RUNDESNTEINER, E. A. Honey, I shrunk the XQuery!: an XML algebra optimization approach, In: Fourth International Workshop on Web Information and Data Management, LcLean, Virginia, 2002, **Proceedings…**, p.15-22.

[Zhou et al. 1996] ZHOU, G., HULL, R. AND KING, R. Generating data integration mediators that use materialization, **Journal of Intelligent Information Systems**, vol. 6, no. 2/3, p. 199-221, 1996.

[Wang et al. 1996] WANG, Y. R., AND STRONG, D. M., Beyond accuracy: what data quality means to data consumers, **Journal on Management of Information Systems**, v. 12, n. 4, p. 5-34, 1996.

[Wiederhold 1992] WIEDERHOLD, G. Mediators in the architecture of future information systems, IEEE Computer, p.38-49, 1992.

[Widom 1995] WIDOM, J. Research problems in data warehouse, In: 4th Int'l Conference on Information and knowledge Management (CIKM), Baltimore, Maryland, 1995, **Proceedings...**, p. 25-30.