Automatic Mapping Generation and Adaptation for XML Data Sources

Xiaohui Xue

THÈSE / PhD THESIS

présentée à

L'UNIVERSITÉ DE VERSAILLES SAINT-QUENTIN-EN-YVELINES

pour obtenir le titre de

DOCTEUR EN SCIENCES

Spécialité

Informatique

soutenue par

Xiaohui XUE

Titre

Génération et adaptation automatiques de mappings pour des sources de données XML

Automatic Mapping Generation and Adaptation for XML Data Sources

(Version Préliminaire)

JURY

Michel Scholl	Professeur des universités, CNAM-Paris	Président du jury, Examinateur
Christine Collet	Professeur des universités, ENSIMAG	Rapporteur
Mohand-Saïd Hacid	Professeur des universités, Université Claude Bernard Lyon 1	Rapporteur
Mokrane Bouzeghoub	Professeur des universités, Université de Versailles	Directeur de thèse
Zoubida Kedad	Maître de conférences, Université de Versailles	Encadrante de thèse, Examinateur

Résumé

L'intégration de l'information fournie par de multiples sources de données hétérogènes est un besoin croissant des systèmes d'information actuels. Dans ce contexte, les besoins des applications sont décrits au moyen d'un schéma cible et la façon dont les instances du schéma cible sont dérivées à partir des sources de données est exprimée par des mappings. Dans cette thèse, nous nous intéressons à la génération automatique de mappings pour un schéma cible XML à partir des sources de données XML ainsi qu'à l'adaptation de ces mappings en cas de changements survenant dans le schéma cible ou dans les sources de données.

Nous proposons une approche automatique de génération de mappings en trois phases : (i) la décomposition du schéma cible en sous-arbres, (ii) la recherche de mappings partiels pour chacun de ces sous-arbres et enfin (iii) la génération de mappings pour l'ensemble du schéma cible à partir de ces mappings partiels. Le résultat de notre approche est un ensemble de mappings ; chacun ayant une sémantique propre. Dans le cas où l'information requise par le schéma cible n'est pas présente dans les sources, aucun mapping ne sera produit. Dans ce cas, nous proposons de relaxer certaines contraintes définies sur le schéma cible pour permettre de générer des mappings. Nous avons également proposé une approche d'adaptation des mappings existants en cas de changement survenant dans un des sources ou dans le schéma cible. Nous avons développé un prototype d'un outil pour supporter notre approche.

Abstract

The integration of information originating from multiple heterogeneous data sources is required by many modern information systems. In this context, the applications' needs are described by a target schema and the way instances of the target schema are derived from the data sources is expressed through mappings. A mapping describes the transformation and the integration of the source instances to conform to the target schema. In this thesis, we address the problem of mapping generation for a XML target schema from multiple XML data sources and the automatic adaptation of these mappings when the target schema or the source schemas evolve.

We propose an automatic generation approach that first decomposes the target schema into subtrees, then defines mappings, called partial mappings, for each of these subtrees, and finally combines these partial mappings to generate the mappings for the whole target schema. Applying our approach creates a set of alternative mappings: all for the given target from the sources. If the information required in the target schema cannot be provided by the sources, then no mapping can be generated. In this case, we propose to relax some constraints of the target schema such that mappings can be generated. We also propose a mapping adaptation approach to keep existing mappings current if some changes occur either in the target schema or in one of the source schemas. We have developed a prototype implementation of a tool to support the generation process.

Table of Contents

List of Figures

Chapter 1. Introduction

1. Context and Motivation

Nowadays, interoperability is an increasingly important issue within many information management contexts. Many modern information systems such as data migration systems, mediation systems, and data warehouses need to import existing data with some particular structure and re-use it in a different format. Building these applications begins with an understanding of how data will be used and viewed; this consists in determining the application's needs, represented by a schema called the ***target schema***. The data required by the application is provided by some data sources. The data exported by each data source is described using a ***source schema***. ***Mappings*** are therefore needed between the target schema and the source schema to express the way instances of the target schema are derived from the instances of the source schemas. That is, they state how the application's data needs are satisfied by the sources.

Consider the example given in Figure 1-1: a data source provides information about professors and students of a university. It exports a source schema that contains information about professors, students, and the teaching relationships between professors and students. Suppose that an application requires the same information in another representation; this application provides a target schema that contains information about persons representing both professors and students and the teaching relationships between persons. A mapping can be defined to specify how instances of the target schema can be derived from the instances of the source schema. Instances of a person are obtained by constructing the union of the professor instances and student instances from the data source. Instances of the teaching relationship in the target schema are obtained from the instances of the same relationship in the data source; the mapping ensures that each teaching instance relates two persons that correspond to a professor and a student related by the data source instance.

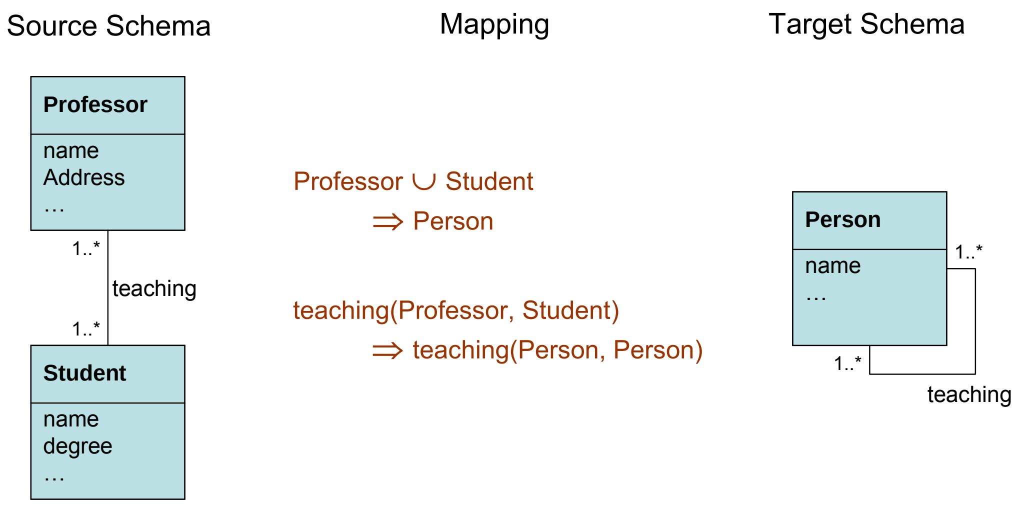


Figure 1-1. One example of a mapping between a target schema and a source schema

Mappings can be used in many different contexts such as data migration systems, data translation systems, wrappers, mediation systems, or data warehouses. A ***data migration system*** [BH, CG04 and LD94] is used to transfer data from a legacy system to a new system. Mappings have to be defined between the schema provided by the legacy system (source schema) and the schema of the new system (target schema). They state the way to populate the new system using the data of the legacy system. A data translation system [ACM97, Bil79 and KA04] manages information exchange between applications. It has to create mappings between the schema of the information provider (source schema) and the schema of the information consumer (target schema) to specify the way information can be exchanged. A wrapper [IS06, LPH00 and THH05] encapsulates a single data source to make it usable in a more convenient fashion than the original source. Therefore it has to specify the mapping between the internal representation (the source schema) and the external representation (target schema). In a mediation system [GPQ97, TRV98 and Wie92] that provides the users a transparent access to heterogeneous data sources, the system defines a global view of these sources (target schema). Mappings either define the target schema in terms of source schemas (in the Global-as-View approach), or define each source schema in terms of the target schema (in the Local-as-View approach). These mappings will be used to rewrite user queries expressed over the target schema into local queries over the sources. In a data warehouse [Inm96 and TS99], mappings are defined between the source schemas and the integrated schema of the system to derive the storage of the data warehouse from the integration of the data sources.

Defining the mappings between the target and the sources is also called ***schema mapping***. It is often considered to be a manual process. To perform this task, the designer must have a thorough understanding of not only all the data sources, but also the semantic links between the sources and the target schema. Especially when there are a large number of data sources such as in mediation systems and data warehouses, the amount of metadata to manage may be very important and makes the manual definition of mapping extremely difficult and time consuming. Thus, it is necessary to provide support for the automatic mapping generation.

Mappings are necessarily dependent on the schemas they relate. However, individual sources are autonomous, freely evolving both their content and their capabilities, or even changing their status between available and unavailable. Likewise, the applications also freely evolve their target schemas to adapt to their new needs. These schema changes lead to continuous obsolescence and re-definitions of the mappings. If the number of the source schemas is small and if the schemas are simple, it is possible to check manually the mappings and to adapt them. But if the number of the schemas is important and if they have complex structures, manually adapting the mappings becomes a difficult task. If some change occurs in the sources or the target schema, one solution is to restart the generation process; but this can be costly, especially when the changes have little impact on the mappings. For example, the renaming of a source element can be propagated in a very simple way in the existing mappings and does not require generating the mapping from scratch.

2. An Overview on Mapping Generation and Adaptation

Given one target schema and several source schemas, mapping generation produces mappings that describe the way instances of the target schema are derived from the instances of the source schemas. If one of the schemas referenced by the mapping evolves, the mapping may become obsolete. Mapping adaptation consists in adapting this obsolete mapping to keep it consistent with the evolved schemas.

Some research approaches [BKS04, MHH00, FW06, SKR01, Zam04, CLL03, PVM02 and FS03] and industrial solutions [AIS, AMF and Sty] have been proposed for automatically or semi-automatically generate mappings. Some of them [BKS04, MHH00 and FW06] consider relational schemas and the others [SKR01, Zam04, CLL03, PVM02, FS03, AIS, AMF and Sty] consider XML schemas. Many of the existing approaches generate mappings over one single source schema. Only two approaches [BKS04 and FS03] generate mappings over several source schemas and the output mappings can express joins between different sources. Some approaches [BKS04, MHH00, SKR01 and CLL03] generate mappings described in a standard query language such as SQL, XQuery or XSLT. The other approaches [FW06, Zam04, PVM02 and FS03] generate mappings expressed in a specific language. Among these existing works, some [SKR01, CLL03 and FS03] still assume that the target schema and the source schemas have homogeneous structures. The existing industrial tools can only generate very simple mappings to derive instances for every construction of the target without considering the structure of the whole schema.

Some approaches [BFK03, LNR02, MP02, VMP04 and YP05] have been proposed to adapt mappings automatically when the related schemas evolve. The approaches proposed in [BFK03 and LNR02] assume relational schemas and mappings expressed in SQL. [VMP04 and YP05] consider nested data models and mappings expressed in a specific language. [MP02] assumes schemas described using an Entity-Relationship like model and mappings are expressed using a series of transformation operations. The approaches proposed in [BFK03 and VMP04] are related to an existing mapping generation methodology and the proposed mapping adaptation can be considered as an incremental execution of this methodology. These approaches assume that all the intermediate results of the mapping generation are provided. They define, for every change they consider, the actions of re-computing the intermediate results and then the final results of the mapping generation process. The approaches presented in [MP02 and YP05] consider that schema changes are expressed using a mapping and the mappings have to be generated using a specific mapping generation approach. Their adaptation consists in combining the original mapping with the mapping expressing the changes to obtain new mappings. The approach proposed in [LNR02] adapts mappings between relational schemas for its considered changes. This adaptation is performed using some information about the source such as the possible joins between source relations and the inclusion relations between different source instances, etc.

All the existing approaches on mapping generation and mapping adaptation use semantic correspondences. Semantic correspondences involve elements of different schemas and state that these elements represent the same concept. For example, a semantic correspondence can be defined between "professor" and "teacher" to state that they represent the same meaning. Transformation functions can be used in semantic correspondences. Generating semantic correspondences is also called ***schema matching***. Providing support to schema matching is a difficult problem and has been an active research topic for a long time [BLN86] and is still an active research area [DLD04, DR02, Gal06, HC06, RB01 and SE05]. The definition of semantic correspondences is beyond the scope of our work.

3. Our Proposal

Only some of the existing mapping generation approaches can produce mappings involving joins between different sources. The approach proposed in [FS03] generates mappings involving inter-source joins for XML schemas. However, it assumes that the target schema and the source schemas have homogeneous structures; this cannot always be the case in distributed and heterogeneous environments. Most of the existing works proposed for mapping adaptation are related to a specific mapping generation approach. They either require providing all the intermediate results of the mapping generation process, or the generation of a mapping to describe the schema evolution. The approach proposed in [LNR02] does not make such an assumption but it requires a large volume of source descriptions and some of them are difficult to obtain such as the list of all possible joins between the sources.

In this thesis, we address the problems of automatic mapping generation and adaptation. We assume that the target and source schemas are described in XML Schema. We generate automatically a set of mappings between the target schema and the source schemas and we adapt existing mappings when the target and source schemas evolve automatically as well. Similarly to the existing approaches, we assume that a set of semantic correspondences is provided between the schemas.

To handle the complexity of mapping generation, our approach comprises three steps:

- decomposing the target schema into a set of subtrees, called ***target subtrees***;
- then finding different ways, called ***partial mappings***, to derive instances of each target subtree from the instances of the source schemas; the partial mapping generation for each target subtree is done independently from the others.
- finally combining the partial mappings of different target subtrees to generate the mappings for the whole schema, called ***target mappings***.

The result of our approach is a set of mappings having different semantics. They are specified in an abstract language and can be translated into another specific language such as XQuery. The generated mappings can involve joins between different sources. Every resulting mapping represents an alternative way to derive instances of the target schema that is different from the others.

If some information required in the target schema can not be found in the data sources, no mapping can be generated (e.g. the target schema asks for "authors" writing computer science books but no source has this information). We adapt our mapping generation approach to handle this case. We propose to change the target schema by relaxing some cardinality constraints or some structural constraints such that a mapping can be found for the new schema. The process is an interactive process where the user is asked to validate the modifications made on the target schema or to choose one relaxation option among the other alternatives.

We also provide an automatic approach to enable the adaptation of mappings that are either generated by a specific mapping generation tool or manually specified by a designer. We consider schemas described in XML Schema [xsd] and mappings expressed using XQuery [xquery]. We adapt the mapping for a given set of changes (e.g. removing a key constraint from a source schema or adding an element in the target schema). The adaptation is done in three steps:

- The original mapping is first checked to see if it is ***affected*** by the change. Not all the changes affect the mapping. For example, a mapping will not be affected by a change that occurred in a schema that is not referenced by the mapping. A mapping can be affected in two cases: either the target schema has changed and the mapping does not satisfy the new needs anymore, or some source resources that are used in the original mapping have been moved or removed.
- The affected mapping is checked to see if it is ***adaptable***, which means there is at least one solution to adapt it to the new schemas. If some new components are added to the target schema, the mapping has to be extended to derive instances for these new components. If some source components used in the mappings are moved or removed, some substitutions have to be found to replace the removed ones.
- If the mapping is adaptable, the process makes use of all the solutions found during the second step to generate ***adapted mappings***.

The result of applying the mapping adaptation approach may create several adapted mappings; every adapted mapping represents an alternative way to adapt the mapping to the new schemas. We propose a set of adaptation algorithms for every type of changes.

Our mapping adaptation approach allows the adaptation of the mappings after changes occurring in the source schemas; this approach does not rely on a specific methodology for mapping generation. Both target schema changes and source schema changes are supported.

4. Outline of the Thesis

The remaining of this thesis is organized in five chapters:

Chapter 2 presents a state of the art on mapping generation and mapping adaptation. It first provides some system contexts in which a mapping can be used. Then, we present the existing works on mapping generation. The principles of these works are first presented. Then we give some discussions over the main features of these approaches such as the characteristics of the input data, the characteristics of the result mappings and the methodology used to generate mappings. We also present the existing works on mapping adaptation. We present the principles of these works and we give some discussions over the main characteristics of these approaches such as the changes considered by the approach, the system cost for adapting mappings, and the generalizability of these approaches.

Through the study of the existing work, we identify some limitations of the existing approaches: no approach can generate mappings for multiple XML schemas such that the input schemas do not have to be homogeneous and the result mappings can involve joins between different sources. No approach adapts mapping without relying on a specific mapping generation approach and without relying on a large volume of user input metadata. We conclude by positioning our work with respect to these existing works.

Chapter 3 describes our mapping generation approach and the associated algorithms; we also present the adaptation of the approach to generate incomplete results when no mapping can be generated for the target schema.

We propose an approach to automatically generate mappings for XML schemas using a set of semantic correspondences. The mapping generation process is composed of three steps. The target schema is first decomposed into several subtrees. Then mappings, called partial mappings, are defined for each of the subtrees. The definition of the partial mappings for each subtree is done independently from the others. Finally, partial mappings of different subtrees are combined to generate the mappings for the whole target schema. The result is a set of different mappings: all for the given target from the sources; every one represents a different semantics. We presented all the algorithms used in the generation process.

If the information required by the target schema cannot be provided by the sources, no mapping can be generated. We present our adaptation of the mapping generation approach to generate incomplete results. The idea is to relax some constraints defined in the target schema in a way that a mapping can be generated for the new target schema. This process requires user interaction to either validate a proposed relaxation or to choose one relaxation manner between all the alternative ones.

Chapter 4 describes our mapping adaptation approach and the associated algorithms. We propose an approach to automatically adapt mappings when schemas evolve. The schemas are XML schemas and the mappings are expressed using XQuery in a particular format that we consider. We consider a set of changes that are either in the source schemas or in the target schema and we adapt the mapping for each of these changes in three steps. We first checks if the original mapping is affected by the change. If the mapping is affected, we checks if it is adaptable with respect to the new schemas; which means if there is at least one solution to adapt it to the new schemas. If the mapping is adaptable, all the adaptation solutions are used to generate mappings for the new schemas. The algorithms used in our mapping adaptation are also presented.

Chapter 5 presents the prototype system that implements our approaches presented in Chapters 3 and 4 and some experimental results. We first describe our prototype implementation for automatic mapping generation and adaptation. This tool allows the execution of the algorithms presented in Chapters 3 and 4 to perform the tasks of mapping generation, incomplete result generation and mapping adaptation. The tool also provides some facilities to allow the use to select the input metatdata, to monitoring the intermediate results and to interact with the system, etc.

The experimentation of the approach is twofold. First, the prototype has been used to generate mappings in different scenarios; this enables to validate the approach. We describe two applications: (i) the ACI-MediaGrid project that proposes a mediation framework to allow transparent access to distributed sources, (ii) an adaptive system for aiding in the generation of mediation queries. We briefly describe each application and we explain how our tool has been used. We also performed some tests for evaluating the performance of the tool. To this end, we consider some scenarios we executed the mapping generation algorithm over each scenario. The test results allow affirming that the tool can be used in actual systems.

Chapter 6 presents conclusions and some research perspectives.

Chapter 2. Mapping Generation and Adaptation: Related Works

1. Introduction

Many applications require the use of existing data, stored in multiple distributed and possibly heterogeneous sources. Considering that the application needs are represented by a target schema, mappings have to be defined to express the way instances of a target schema are derived from the instances of the source schemas. Such mappings can be used in many different contexts such as ***data migration systems***, ***data translation systems***, ***wrappers***, ***mediation systems***, or ***data warehouses***.

Mappings can be defined between one source and one target schema, as it is the case for data migration systems, data translation systems and wrappers:

- data migration is necessary when an organization decides to use a new database management system that is incompatible with the current one; a data migration system [BH, CG04 and LD94] is used in this case to transfer data from the legacy system to the new one.
- data translation refers to the information exchange between applications; a data translation system [ACM97, Bil79 and KA04] performs the transformation of data from its own application's representation to another;
- A wrapper [IS06, LPH00 and THH05] encapsulates a single data source to make it usable in a more convenient fashion than the original unwrapped source. It specifies the transformer of data from the internal representation to the external representation.

In the design of all these systems, mappings define the manner of transferring data from one representation (the source schema) to another (the target schema); the two schemas are expressed in the same model (e.g. relational model, XML model). If the schemas are expressed in different models, some approaches [ACB06, MRB03 and SKS01] can be used to specify the model transformation between the schemas (e.g. transforming a schema in a relational model to an object representation). Figure 2-1 is inspired by [THH05] to illustrate the general process of the transformation in these systems. Consider a source schema *S1* and a target schema *S4* expressed in different models. For specifying the transformation of the instances of *S1* to *S4*, the system may have to specify: (i) first the model transformation from *S1* to its equivalent schema *S1* that is expressed using the internal model of the system, (ii) then the mapping from *S2* to *S3* that is the target schema expressed using the internal model of the system, (iii) and finally the model transformation from *S3* to *S4*. If the internal model of the system is the same as the source model, the transformation from *S1* to *S2* is not needed and the mapping is specified directly from *S1* to *S3*. If the internal model of the system is the same as the source model, the transformation from *S3* to *S4* is avoided and the mapping is specified directly from *S2* to *S4*. If all the three data models are the same, there is no model transformation and only the mapping need to be specified from *S1* to *S4*.

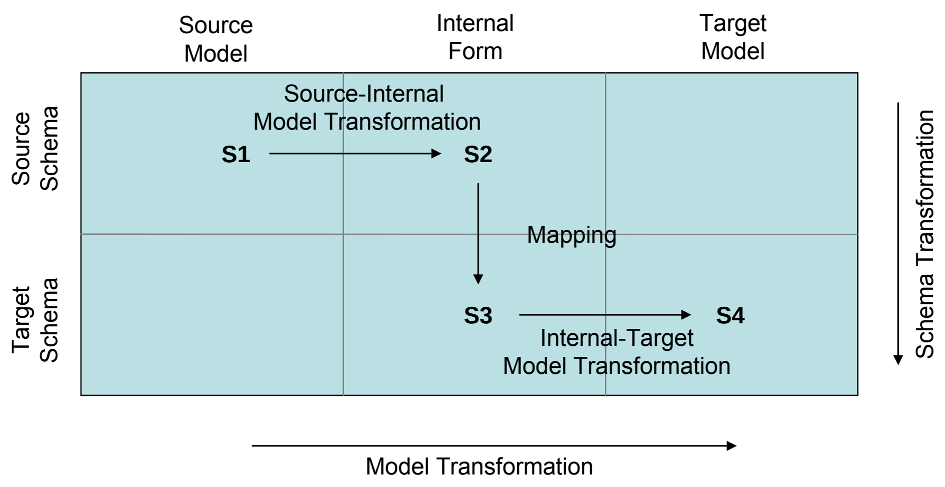


Figure 2-1. The general process of the transformation specification from a source to a target in a data migration system, a data translation system or a wrapper

Beside the previous use contexts, mappings can also be defined between several sources and a target schema as it is the case for mediation systems and data warehouses:

- mediation systems [GPQ97, TRV98 and Wie92] are systems that allow a transparent access to distributed and heterogeneous sources by providing the user a global and uniform view of the sources as well as the mechanisms that rewrite user queries over the global view into source queries and returns integrated result.
- data warehouses [Inm96 and TS99] collects, integrates and stores an organization's data with the aim of producing accurate and timely management of information and support for analysis techniques. They provide an integrated schema and data is computerized from the multiple sources to populate the integrated schema.

These systems are also called ***data integration systems*** since they all integrate data sources into a global representation. During the design of these systems, the global schema can be elaborated independently from the source schemas. Mappings are then defined to specify the way of deriving instances of the global schema from the sources.

There exist also other research issues in the data integration systems. One of them is how to resolve errors and inconsistencies between data from different sources (e.g. the dates can be expressed in one source with the format DD-MM-YYYY and in another source with the format MMDDYY). Resolving such conflict is known as ***data cleaning*** and it has been the topics for many research works (a state of the art on data cleaning is provided in [Sou05]). In mediation systems, another research topic consists in specifying the algorithm of ***user query rewriting*** into local queries over the sources. In data warehouse, one of open problem is how to adapt the instances in the data warehouse current to the source instances. For example, when some objects (e.g. tuples in a relational source) are removed from the sources, the instances inside the data warehouse needed to be maintained by removing the same objects. This problem is known as ***view maintenance*** [AMP98 and KR02].

Manually defining the mappings is a difficult task, especially for data integration systems in which there may be a large number of data sources. The designer must have a thorough understanding of not only all the data sources, but also the semantic links between the sources and the target schema. Thus, providing a computer support to help generating these mappings is necessary. For this purpose, many research approaches [BKS04, MHH00, FW06, SKR01, Zam04, CLL03, PVM02 and FS03] as well as industrial solutions [AIS, AMF and Sty] have been proposed for automatic or semi-automatic generating mappings.

Mappings are necessarily dependent on the schemas they relate; data sources are freely evolving both their content and their capacities and the target schema may also evolve according to the evolution of the users' needs. When one of these schemas evolves, the existing mappings may become obsolete and need to be redefined. The mapping definition done during the design time of an information system is usually performed once for all the life cycle of the system. On the opposite, , schema evolutions may require the continuous redefinition of the mappings at runtime; sometimes when a user query is waiting for the results and its rewriting recognizes an inconsistent between the mapping and schemas. Redefining mappings in this context is more considered to be a costly process. Moreover, the original mappings also embody certain preferences with respect to the other alternative mappings, so that the re-definition of new mappings may ignore these preferences. For these reasons, mapping adaptation making use of the original mapping and adapting it to the new schemas seems to be more promising in terms of performance and of the resulting mapping semantics. There are some existing solutions [BFK03, FP04, FS04, LNR02, MP02, VMP04 and YP05] for mapping adaptation.

All the existing approaches on mapping generation and mapping adaptation take a set of semantic correspondences as input. The definition of these correspondences is known as ***schema matching***. Providing a support to schema matching is a difficult problem and the research domain was being devoted to it from years ago [BLN86]; while the problem still stays to be an actuality today [DLD04, DR02, Gal06, HC06, RB01 and SE05]. We do not address the problem of schema matching in this thesis and we omit the presentation of the existing approaches.

In the remainder of the chapter, we present a state of the art on mapping generate in Section 2. Section 3 will describe these existing approaches as well as discuss their solutions. Section 4 concludes the chapter.

2. Mapping Generation

The problem of mapping generation, also known as schema mapping, consists in generating the way instances of a given target schema are derived from the instances of a set of data sources.

Manual mapping generation is difficult to achieve, especially in in presence of many source schemas. The need of managing a large number of metadata about the schemas and all the links between these schemas makes manual mapping generation a difficult task. For this purpose, many solutions have been proposed to semi-automatic or automatic mapping generation in both research and industry. Since 1999, many research approaches have been proposed to generate mappings: [BKS04, MHH00, FW06, SKR01, Zam04, CLL03, PVM02 and FS03]. Some industrial tools have also been developed to automatically generate mappings such as Adeptia Integration Server, Altova MapForce and Stylus.

Among the research approaches for mapping generation, we distinguish between the generation approaches for relational schemas [BKS04, MHH00 and FW06] and the ones for XML representation [SKR01, Zam04, CLL03, PVM02 and FS03]. The industrial solutions all consider XML representations.

In this section, we describe the principle of the different mapping generation approaches and tools. The approaches generating mappings for relational schemas are first presented in Section 2.1. Section 2.2 describes mapping generation approaches for XML representation. The industrial solutions will be described in Section 2.3. Section 2.4 discusses the existing works and Section 2.5 presents their limitations. Section 2.6 presents an overview of our proposal and compares it to the related works.

2.1. Mapping Generation for Relational Schemas

Three approaches, Kedad and Bouzeghoub'99 [KB99, BKS04], Clio'00 [MHH00] and TUPELO [FW06], have been proposed to generate mappings between target and source schemas expressed in a relational model that are described in the following.

Kedad and Bouzeghoub'99 [KB99, BKS04]

This approach has been proposed in the context of mediation systems in which target schemas are called mediation schemas and mappings are called mediation queries. It considers that both mediation and source schemas are relational schemas and mediation schemas are defined by domain experts independently from the sources. The objective of this approach is to help users by generating a set of candidate mediation queries to derive instances of the mediation schema from the instances of the source schemas.

Mediation queries are generated for every relation of the mediation schema. The generation process comprises three steps: (i) searching for the sources that are relevant to the mediation relation; (ii) identifying candidate operations; (iii) defining mediation queries.

The step of searching for relevant sources consists in finding all source relations that can contribute to the computation of the mediation relation. A source relation *Si* is contributive if it includes some attributes of the mediation relation. In this case, a *mapping relation* is extracted from it; the mapping relation contains all the common attributes between the mediation relation and *Si*. The primary key and foreign keys of *Si* are added into the mapping relation. Consider the following example in which there is one mediation relation *Rm(#K,A,B,C)* and four source relations *S1(#K,A,@X,Y)*, *S2(#X,B,Z)*, *S3(#B,C,W)* and *S4(#B,C,U)*. Primary key attributes are prefixed by # and foreign key attributes are prefixed by @. In this example, four mapping relations are obtained from *S1*, *S2*, *S3* and *S4*: *T1(#K,A,@X)*, *T2(#X,B)*, *T3(#B,C)* and *T4(#B,C)*.

The step of candidate operation identification searches for possible joins between mapping relations. A join is possible between two mapping relations *R1* and *R2* either (i) if *R1* and *R2* are in the same source and there is an explicit referential constraint between them or (ii) if *R1* and *R2* are in different sources and the primary key of one has an equivalent attribute in the other. Figure 2-2 shows an example of possible operations for the example. The join 1 is possible between *T1* and *T2* because there is a referential constraint from *T1* to *T2* through the attribute *X*. The join 2 is possible between *T2* and *T3* because the attribute B exists in both *T2* and *T3* and *B* is defined to be a key in *T3*.

There is not always a candidate operation between two mapping relations following the previous rule. However, they might be joined through a third relation. Some relations that contain only primary keys and foreign keys and that have no common attribute with the mediation relation are considered by the algorithm to make possible joins between mapping relations; they are called *transition relations*. For example, consider the two mapping relations *T5(#D, E)* and *T6(#F, G)*. There is no possible join between them. Suppose that there is another source relation *S7(#F, @D, H)* and none of *F*, *D* and *H* is in the mediation relation, then *S7* can be used to join *T5* and *T6*: a join between *T5* and *T7* with the predicate over *D* and a join between *T6* and *T7* with the predicate over F. A transition relation is generated from *S7*; it contains only foreign keys and primary keys: *T7(#F, @D)*. Both mapping relations and transition relations are called *relevant relations*.

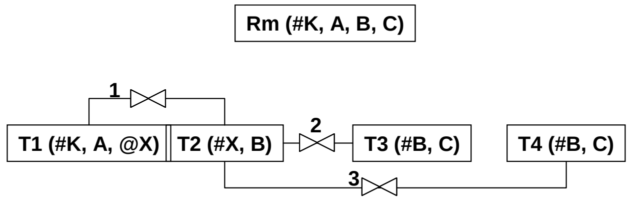


Figure 2-2. Example of operation graph

Relevant relations and the joins between them can be represented by a graph (called *operation graph*) in which every node is a relevant relation and every edge is a join. Over the operation graph, a mediation query is defined from a *computation path*, which is a connected, acyclic sub-graph that involves all the attributes of the mediation relation. Defining mediation queries consists in enumerating all the computation paths from the operation graph. In the example of Figure 2-2, C1 = (1, 3) and C2 = (1, 2) are two computation paths. Their corresponding mediation queries are respectively:

$$E1 = \Pi_{K,A,B,C}[(\Pi_{K,A,X}S1) \bowtie (\Pi_{X,B}S2) \bowtie (\Pi_{B,C}S4)];$$
$$E2 = \Pi_{K,A,B,C}[(\Pi_{K,A,X}S1) \bowtie (\Pi_{X,B}S2) \bowtie (\Pi_{B,C}S3)].$$

Set-based operations such as union, difference, intersection can be used over existing mediation queries to derive new mediation queries. For example, a third mediation query can be generated by applying a union to E1 and E2:

$$E3 = \Pi_{K,A,B,C}[(\Pi_{K,A,X}S1) \bowtie (\Pi_{X,B}S2) \bowtie (\Pi_{B,C}S4)] \cup \Pi_{K,A,B,C}[(\Pi_{K,A,X}S1) \bowtie (\Pi_{X,B}S2) \bowtie (\Pi_{B,C}S3)].$$

Clio'00 [MHH00]

This approach is the first result of the Clio project to generate mappings between relational schemas. Given one source schema and one target schema, the approach generates one mapping semi-automatically.

This approach assumes the existence of a set of value correspondences. Each value correspondence is a function defining how a value (or combination of values) from a source database can be used to form a value in the target. Consider the target and source schemas in Figure 2-3, then the following correspondence *f1* expresses how *Personal(Sal)* is formed from the product of the values of *PayRate(HrRate)* and *WorksOn(Hrs)* attributes:

```
f1:   SELECT      P.HrRate * W.Hrs
      FROM        PayRate P, WorksOn W
      WHERE       P.Rank=W.ProjRank
```

All the value correspondences are shown in Figure 2-3.

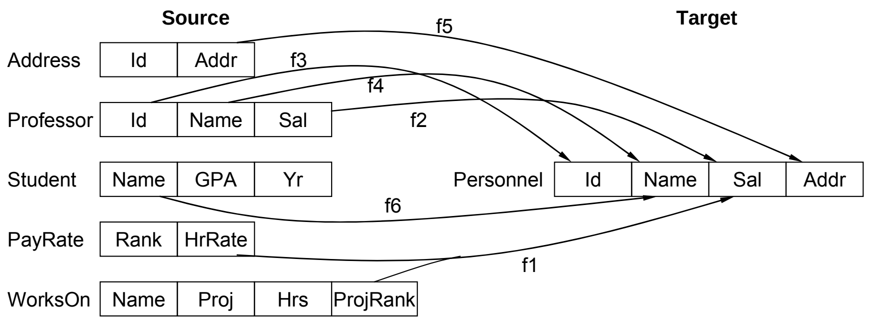


Figure 2-3. Example of target and source relations in Clio'00

Mappings are generated from value correspondences. The algorithm of mapping generation is divided into four phases. The value correspondences are first partitioned into sets such that every one contains at most one correspondence per attribute of T. They are called *potential candidate set*. Each potential candidate set represents a different possible way of mapping the attributes in the target relation. For the example of Figure 2-3, the followings potential candidate sets are listed below. There is first of all a potential candidate set for every correspondence (the first row). Then all possible combinations of the correspondences are considered, every one contains zero or one correspondence for each target element. Some potential candidate sets containing two and three correspondences are shown at respectively the second and the third row. The fourth row shows the potential candidate sets that contain correspondences for each of the four elements of the target schema.

{f1}, {f2}, {f3}, {f4}, {f5}, {f6}, …
{f2, f3}, {f2, f4}, {f2, f5}, {f2, f6}, {f1, f3}, {f1, f4}, {f1, f5}, {f1, f6} …
{f2,f3,f4}, {f2,f4,f5}, {f2,f3,f5}, {f2,f3,f6}, {f1,f3,f4}, {f1,f4,f5}, {f1,f3,f5}, {f1,f3,f6} …
{f2, f3, f4, f5}, {f1, f3, f4, f5}, {f2, f3, f6, f5}, {f1, f3, f6, f5}.

Consider the potential candidate sets. For every one involving more than one source relation, the system checks if there is one relation involved in the set and not related to the others through a foreign key; if such relation is found, the potential candidate set is removed. Potential candidate sets surviving this pruning are *candidate sets*. Assuming that *WorksOn(Name)* is a foreign key of *Student(Name)* and *Address(Id)* is a foreign key of *Professor(Id)*, they we get the following candidate sets from the previous potential candidate sets:

{f1}, {f2}, {f3},{f4},{f5},{f6}, {f2, f3}, {f2, f4}, {f2, f5}, {f3, f4}, {f3, f5}, {f4, f5}, {f1, f6},
{f2, f3, f4}, {f2, f4, f5}, {f2, f3, f5}, {f3, f4, f5}, {f2, f3, f4, f5},

A subset *C* of the candidate sets is generated to cover all the input value correspondences (that is, every value correspondence appears at least once in a candidate set in *C*); it is called a cover. In a cover, one correspondence can participate in multiple candidate sets, but there is no candidate set such that if it is removed, *C* is still a cover. If there is more than one cover, Clio ranks them in reverse order with respect to the number of candidate sets in the cover. In the previous example, the cover that has less candidate sets is:

{f2, f3, f4, f5}, {f1, f6}.

The final step is to build the query from the selected cover. For each candidate set in the cover, a candidate SQL query is generated. Then all SQL queries from the selected cover are combined into one query using the multiset UNION ALL, and this constitutes the resulting mapping.

TUPELO [FW06]

TUPELO generates mappings to derive instances of a target schema from a source schema where both schemas are expressed in the relational model. The approach considers a set of pre-defined transformation operators and uses heuristics to explore the full searching space to find mappings.

The approach makes use of operators defined in FIRA [WR05] such as dropping a column from a relation, or transforming the values of a column *A* into column names with the values of the column computed from the column *B*. The approach also considers complex functions provided by users.

TUPELO is based on the Rosetta stone principle, that is, it considers a set of critical instances *s* and *t* (inputted by users) for the source and target schemas respectively. Data mapping is considered to be a full exploration of the transformation space for the considered transformation operators on the source instance s. The search terminates successfully when the target instance t is located in this space.

Heuristics are used to reduce the search space. The approach uses an evaluation function *f* for ranking a search state *x*: *f(x) = g(x)+h(x)* where *g(x)* is the number of transformations applied to the start state to get to state *x* and *h(x)* estimates the distance from *x* to *t*. The authors discuss different heuristics to estimate the distance from *x* to *t* such as set-based similarity heuristics, the Levenshtein heuristic, or Euclidean heuristics. At each step in the exploration of the search space, TUPELO compares *f(x)* values of different states and selectively searches the space based on the rankings.

2.2. Mapping Generation for XML Representations

Other approaches have been proposed to generate mappings for target and source schemas in an XML model such as Xtra [SKR01], AutoMed [Zam04], ORA-SS [CLL03], Clio'02 [PVM02] and Farias Lóscio and Salgado'03 [FS03]; they are described in the following.

Xtra [SKR01]

Xtra also generate mappings between one source schema and one target schema. Both schemas are described by DTDs and the mapping generated for the schemas is expressed in XSLT.

DTD schemas are modeled as a tree in Xtra. Four kinds of nodes are possible in Xtra trees: element node, attribute nodes, list nodes (each list node indicates how its children are composed, that is, by sequence < , > or by choice < | >), and quantifier nodes (each quantifier node represents the cardinality of its children with respect to its parent which is one or more < + >, zero or more < * > or zero or one < ? >). Figure 2-4 shows two Xtra DTD trees.

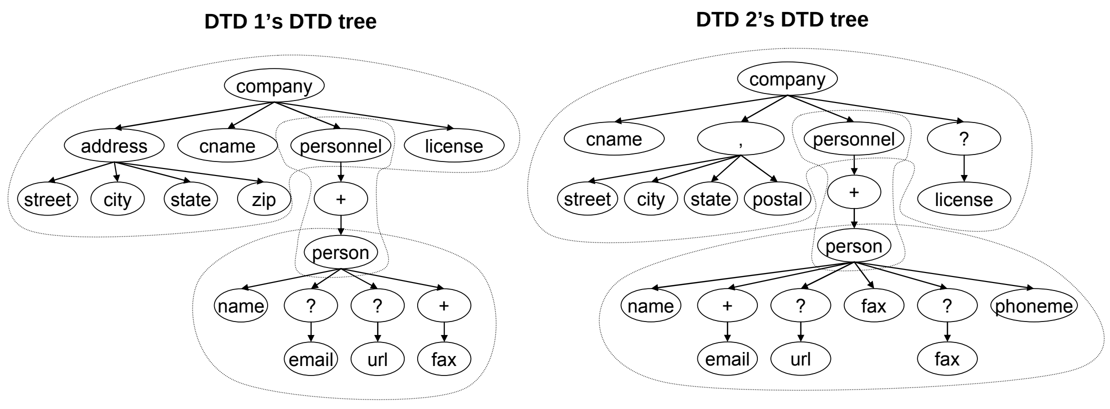


Figure 2-4. Two Xtra DTD trees

The schema transformation process proposed by Xtra consists of three steps. It first generates possible transformation sequences to transform one schema to another using a set of transformation operators. Then it evaluates these operations using a pre-defined cost model and chooses one from all the possible sequences. Finally this operation sequence is used to generate an XSLT transformation script.

A set of transformation operators describes all possible tree transformations considered by Xtra, some of which are:

- *add(T, n)*: add a subtree T under node n.
- *delete(T)*: delete subtree T.
- *remove(n)*: remove node *n* which is a quantifier or a sequence list node. After the removal, all *n*'s children become the children of the parent of *n*.
- *insert(n)*: insert a new node *n* under *p* with *n* a quantifier node or a sequence list node and move a subset of *p*'s children to become *n*'s children.

– *relabel(n, l, l')*: change *n*'s original label *l* to *l'*.

Consider the two DTD trees shown in Figure 2-4. One possible transformation from DTD 1 to DTD 2 consists in first deleting the subtree having the root < *address* > and then adding a subtree having the root < , > and the nodes < *street* >, < *city* >, < *state* > and < *postal* >. Another possible transformation consists in re-labeling < *address* > to < , > and re-labeling < *zip* > to < *postal* >.

A cost model is defined based on data quality, is represented by the number of nodes involved in a transformation sequence. This cost model is used to rank transformation sequences. For the two previous possible transformation sequences concerning addresses, the first one removes a subtree of five nodes and adds a new subtree of five nodes; it involves ten nodes. The second sequence re-labels two nodes. Because there are ten nodes involved for the first sequence while only two nodes are involved for the second one, the data quality with the first transformation sequence is less than with the second one. The latter is therefore preferred. However, argue that the cost model might be different for different situations; for example, a cost model based on the data freshness might result in a different ranking.

To perform the schema transformation, DTD trees are decomposed into subtrees: each subtree has the root *r* such that *r* is not a leaf and it has the same name with a node in another schema. If the root of a subtree has a parent node, then this root is also included in the subtree containing its parent. Figure 2-4 shows the subtrees of the two schemas. There is the same number of subtrees in the two schemas. For each pair of subtrees of the two schemas having the root of the same name, the system searches for all possible transformation sequences of the two subtrees and selects the best one based on the cost model. Once the transformations of all the pairs of subtrees are obtained, they are translated to an XSLT script that represents the mapping to transform the source XML documents into the target format.

AutoMed [Zam'04]

This approach generates transformations (mappings) from one schema and a source schema within the AutoMed system. The schemas of the XML data sources can be DTD or XML schemas that are represented by *XML DataSource Schema* in AutoMed. Four constructs are defined in XML DataSource Schema: elements, attributes, parent-child relationships between elements, and texts. Figure 2-5 shows an example of schema transformation in AutoMed between two schemas S_1 and S_2. Both schemas contain the four constructs. Elements are represented by rectangles and attributes are represented by ellipsis. To solve the problem of multiple elements of the same schema having the same name, every element or attribute is followed by a unique number in the schema. Texts are represented by *PCData* constructs. Parent-child relationships between elements are represented by edges and orders are specified for edges having the same parent.

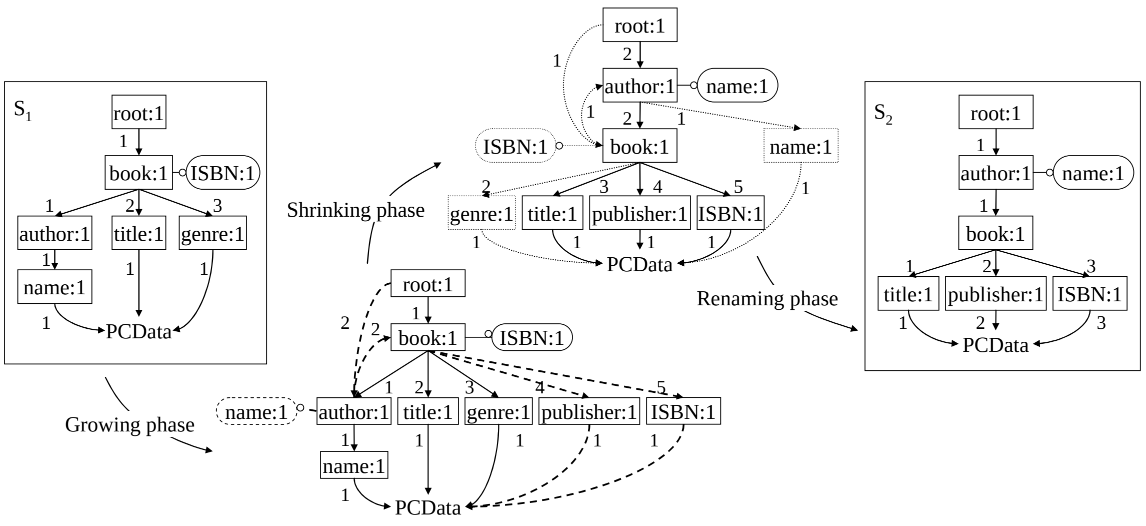


Figure 2-5. Schema transformation in AutoMed

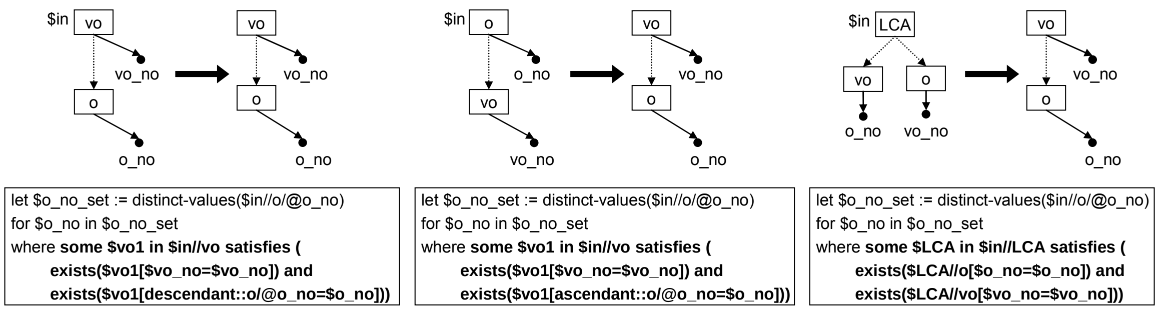
The transformation in AutoMed is described by a sequence of *transformation pathways*: each one expresses a change like adding an element or removing an attribute. Generating the transformation pathways from schema *S1* to schema *S2* consists in three phases: (i) the growing phase that adds to *S1* the constructs that are in *S2* but not in *S1*; (ii) the shrinking phase that removes from the result of the growing phase those constructs that are not in *S2*; (iii) the renaming phase that renames elements and edges in the result of the shrinking phase. All transformations are expressed using pre-defined transformation operators. The sequence of all the transformations in these three phases forms the transformation pathways from *S1* to *S2*. Figure 2-5 shows the three phases in the transformation from *S1* to *S2*.

During the growing phase, *S2* is browsed in a depth-first order. For every element *e* of *S2*, the algorithm processes sequentially the element e itself, its attributes and its text. The algorithm first processes the element *e*. Consider the parent of *e* in *S2* being *p*, and *p* already exists in *S2* when processing *e*. If the element *e* is not in *S1*, it is added to S1 as a child of *p*. If an attribute *a* in *S1* is equivalent to *e*, the process marks that *e*'s data is obtained from *a*. In the case that *e* is in *S1* but its parent in S1 is not *p*, *e* is restructured to be a child of *p*. Once *e* is processed, every attribute *a* of *e* in *S2* that does not exist as an attribute of *e* in *S1* is inserted. If *a* is found at another place of *S1*, an expression will be added to *a* to note how data of *a* is found from *S1*. After adding attributes of *e*, if *e* is linked to the PCData construct in *S2* but the same link is not present in *S1*, this link is added in *S1*. The shrinking and renaming phases are straightforward. The shrinking phase browses the result of the growing phase and removes all the constructs not present in *S2*. The renaming phase renames constructs and re-defines the order of the edges.

ORA-SS [CLL03]

This approach is done in the ORA-SS project. Given an integrated view (global schema) being defined as the integration of the source schemas [YLL03], the approach generates a view definition (mappings) for the integrated views from each source schema. The algorithm generates view definitions that are represented by XQuery queries to express the restructuring of instances of the source to the integrated schema.

The source and target schemas are all modeled by the ORA-SS (Object-Relationship-Attribute) model [DWL00]. The model comprises three basic concepts: object classes, relationship types and attributes. Figure 2-6 gives example of an ORA-SS model where object classes are presented by rectangles and key attributes of object classes are denoted by filled circles. Referential constraints are not supported.



```
let $o_no_set := distinct-values($in//o/@o_no)
for $o_no in $o_no_set
where some $vo1 in $in//vo satisfies (
      exists($vo1[$vo_no=$vo_no]) and
      exists($vo1[descendant::o/@o_no=$o_no]))
```

```
let $o_no_set := distinct-values($in//o/@o_no)
for $o_no in $o_no_set
where some $vo1 in $in//vo satisfies (
      exists($vo1[$vo_no=$vo_no]) and
      exists($vo1[ascendant::o/@o_no=$o_no]))
```

```
let $o_no_set := distinct-values($in//o/@o_no)
for $o_no in $o_no_set
where some $LCA in $in//LCA satisfies (
      exists($LCA//o[$o_no=$o_no]) and
      exists($LCA//vo[$vo_no=$vo_no]))
```

(a) *vo* is an ancestor of *o* in the source schema

(b) *vo* is an descendant of *o* in the source schema

(c) *vo* is neither ascendant nor descendant of *o* in the source schema and their lowest common ancestor in the source is not an ascendant of o in the integrated schema

Figure 2-6. Examples of schema transformation in ORA-SS

The algorithm generates the definition of each object class individually and then combines them all together according to the tree structure of the view. The definition of an object class *o* comprises of a `FLWR` expression, that consists of `for`, `let`, `where` and `returns` clauses: the `where` clause restricts the data instances represented by *o*; the `for` clause binds a variable to iterate over each distinct key value of *o* that is qualified by the `where` clause; the `return` clause constructs the instances of *o*, including the attributes associated to *o*.

Generating `for` and `return` clauses is straightforward for an object class but generating `where` clauses is more difficult. Ways to derive the `where` clause for an object class *o* from a source depends on how its ascendants in the integrated view are represented in the source as well as its relationship with *o* in the source. Given an object class *o* and an ascendant *vo* of *o* in the integrated schema, there are three kinds of relationships between *vo* and *o* in the source: *vo* is ascendant of *o*; *vo* is descendant of *o*; or *vo* is neither descendant nor ascendant of *o* and their lowest common ascendant in the source is not an ascendant of *o* in the integrated schema. A pattern is used to generate the `where` clause for each of these three possibilities and they are shown in Figure 2-6 (a, b and c). If *vo* and *o* are siblings and their lowest common ascendant in the source is an ascendant of *o* in the integrated schema, the relationship between *vo* and *o* is expressed using the relationship between this lower common ascendant and *o*.

The algorithm browses the integrated schema from top to bottom. For every object class *o*, it generates all `where` clauses with respect to all its ascendants being in one of the three cases. The result gives the view definition of the integrated schema from the source schema. However, we consider that it is sometimes not necessary to generate `where` clauses with respect to all the ascendants. In a target schema, if an object class *o* has the parent *p* that has parent *a*, it is not necessary to generate a `where` clause for *o* with respect to *a*. There is a `where` clause for *p* with respect to *a* and a `where` clause for *o* with respect to *p*, instances of *o* are then grouped with respect to instances of *a* through the `FLWR` expression of *p*.

Clio'02 [PVM02]

Besides the approach proposed to generate mappings for relational schemas [MHH00], the Clio project have proposed another approach to generate mappings between schemas that are in either relational or XML model. Mappings are generated between one source schema and one target schema. Given a set of correspondences, the approach generates a set of alternative mappings.

Clio'02 uses a nested relational data model to represent both relational and semi-structured schemas. Figure 2-7 shows a relational source schema *expenseDB* and an XML target schema *statDB*. Foreign keys are specified between elements of the same schema and correspondences existing between the two schemas are represented by arrows.

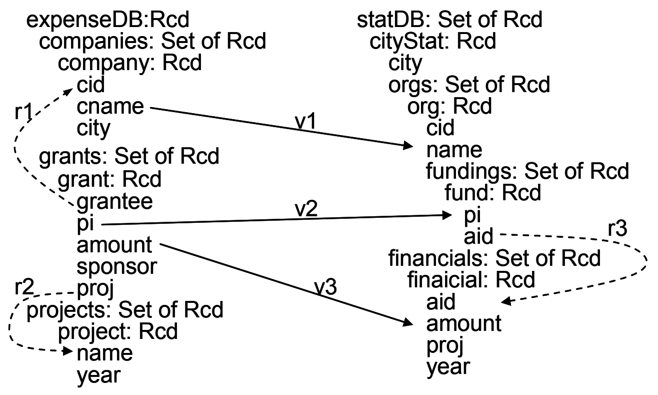


Figure 2-7. Example target and source schemas in Clio'02

A set of *primary paths* is created for each schema to denote its tree structure such that each one is the set of all elements found on a path from the root to any intermediate node or leaf in this tree. The three primary paths of *expenseDB* (*S1* to *S3*) and the four primary paths of *statDB* (*T1* to *T4*) are as follows:

*S1: select * from c in expenseDB.companies*
*S2: select * from g in expenseDB.grants*
*S3: select * from p in expenseDB.projects*
*T1: select * from s in statDB*
*T2: select * from s in statDB, o in s.cityStat.orgs*
*T3: select * from s in statDB, o in s.cityStat.orgs, f in o.org.fundings*
*T4: select * from s in statDB, f' in s.cityStat.financials*

The primary paths are combined to form logical relationships. This process is done using the *chase* method [PT99] that expands primary paths using foreign keys. The logical relationships for *expenseDB* and *statDB* are shown in Figure 2-8. The primary paths *S1*, *S3*, *T1*, *T2* and *T4* cannot be chased because there is no foreign key originating from them; they form logical relationships by themselves (*A1*, *A3*, *B1*, *B2*, *B4* respectively). The primary paths *S2* can be chased using the two foreign keys in *expenseDB* to produce the logical relationship *A2*. The primary paths *T3* can be chased using the foreign key of *statDB* to produce the logical relationship *B3*.

```
A1: select  c.company.cid, c.company.cname, c.company.city
      from  c in expenseDB.companies
A2: select  c.company.cid, c.company.cname, c.company.city,
            g.grant.pi, g.grant.amount, g.grant.sponsor,
            g.grant.proj, g.project, year
      from  g in expenseDB.grants, c in expenseDB.companies,
            p in expenseDB.projects
     where  c.company.cid=g.grant.grantee and
            p.project.name=g.grant.proj
A3: select  p.project.name, p.project.year
      from  p in expenseDB.projects
```

```
B1: select  s.cityStat.city from s in statDB
B2: select  s.cityStat.city, o.org.cid, o.org.name
      from  s in statDB, o in s.cityStat.orgs
B3: select  s.cityStat.city, o.org.cid, o.org.name, f.fund.pi,
            f.fund.aid, f'.financial.amount, f'.financial.proj,
            f'.financial.year
      from  s in statDB, o in s.cityStat.orgs, f in o.org.fundings,
            f' in s.cityStat.orgs
     where  f'.financial.aid=f.fund.aid
B4: select  s.cityStat.city, f'.financial.amount, f'.financial.proj,
            f'.financial.year
      from  s in statDB, f' in s.cityStat.orgs
```

Figure 2-8. The logical relations in *expenseDB*, *statDB*

Consider every logical relationship in the source schema with every logical relationship in the target schema. A logical pair is formed from these two logical relationships if there are correspondences between their elements. For example, the following logical relationship *v12* is formed by *A2* and *B2*: the `for-where` clause represents *A2*; the `exists` clause represents *B2*; and the `where` clause contains the correspondences *v1* and *v2* that are between elements of *A2* and *B2*.

```
v12: for      g in expenseDB.grants, c in expenseDB.companies,
              p in expenseDB.projects
     where    c.company.cid=g.grant.grantee and p.project.name=g.grant.proj
     exists   s in statDB, o in s.cityStat.orgs, f in o.org.fundings, f' in s.cityStat.orgs
     where    f'.financial.aid=f.fund.aid and o.org.name= c.company.cname
              and f.fund.pi= g.grant.pi
```

A set of logical pairs are generated from all pairs of the source logical relationships and the target logical relationships satisfying the condition. This is the mapping solution of the approach introduced by Clio'02. The two logical pairs in the same result may derive instances for the same portion of the target because logical relationships from the same source may be overlapping. A rewriting algorithm [YP04] is then proposed to rewrite user queries using these mappings into source queries.

Two other approaches [ABM05a, ABM05b] directly inspired by the Clio project have been proposed for generating mappings between a conceptual model and a relational table [ABM05a] or between a conceptual model and an XML schema [ABM05b]. The adaptation consists in using a chase-like method to generate logical relationships over variables of the conceptual model. Then referential constraints of the relational schema or XML schema tree structure are used to check the generated logical relationships from which the relevant ones are selected. All these mappings are expressed in a specific format, similar to the ones in [PVM02].

Farias Lóscio and Salgado'03 [FS03]

This approach is proposed in the context of mediation systems in which mediation schemas and source schemas are all expressed through an XML schema. The approach generates a set of candidate mediation queries to define alternative ways for deriving instances of the mediation schema from the sources.

The proposed process first translates the schemas into an ER liked model, called *X-Entity* [FSG03]; then mediation queries are generated for each entity of the mediation schema [FS03]. The second step of the approach is inspired from the ones discussed earlier [KB99, BKS04].

The X-Entity model [FSG03] consists in a set of entity types and a set of containment relationship types. Each entity type represents an entity composed by attributes. Every attribute is associated with a domain and a cardinality to specify the minimum and the maximum number of instances of the attribute that can be related to an instance of the entity. Figure 2-9 shows one mediation schema and three source schemas in the X-Entity model. Rectangles represent entities and ellipses represent attributes. Dotte0d lines are used to connect optional attributes and thick lines are used to connect required attributes. Multivalued attributes are represented by double ellipses. Containment relationships between entity types specify that each instance of one entity contains instances of the other. They are characterized by a cardinality specifying the minimum and maximum number of instances of the contained entity in the containing entity.

An X-Entity diagram can be derived automatically from an XML schema. Every element in the schema having the type defined as `complexType` is represented as an entity. Other elements and attributes are represented by attributes of the entity that corresponds to their first ascendant. For every element *e1* of the type `complexType` and its first ascendant *e2* of the type `complexType`, there is a containment relationship between the entity of *e1* and the entity of *e2*.

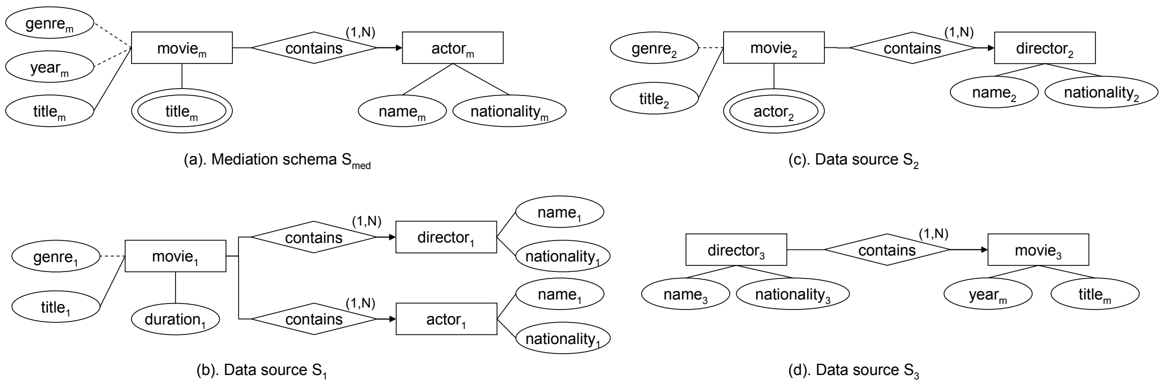


Figure 2-9. One mediation schema and three source schemas in the X-Entity model

Mediation queries are generated for every entity of the mediation schema. The process consists in three steps: (i) selection of relevant source entities that potentially allow computing of the mediation entity; (ii) identification of possible operators to apply between different relevant source entities; (iii) generation of all possible queries from the selected source entities and operators.

Given a mediation entity E_m, a source entity E_s is relevant to the computation of E_m if E_s and E_m have equivalent names. *Mapping views* V({X_1, .., X_n}{Y_1, .., Y_n}) are then defined to specify how E_m can be computed from E_s. Every attribute X_i is equivalent to an attribute in E_m. It either belongs to E_s or belongs to another source entity that is related to E_s by a path of containment relationships. For every relationship from E_m to another mediation entity E'_m, Y_i is either a relationship between two source entities equivalent to E_m and E'_m or a path of relationships between the two source entities respectively. For the example of Figure 2-9, there are three mapping views for the mediation entity *movie$_m$* from the three source entities *movie$_1$*, *movie$_2$* and *movie$_3$* respectively

V_{movie1}({title$_1$,genre$_1$, movie$_1$.movie$_1$_director$_1$.director$_1$.name$_1$}{movie$_1$_director$_1$});
V_{movie2}({title$_2$,genre$_2$, movie$_2$.movie$_2$_director$_2$.director$_2$.name$_2$}{});
V_{movie3}({title$_3$,genre$_1$, (director$_3$.director$_3$_movie$_3$.movie$_3$)$^{-1}$.name$_3$, year$_3$}{});

In V_{movie1}, the expression *movie$_1$.movie$_1$_director$_1$.director$_1$.name$_1$* specifies the attribute *name$_1$* related to *movie$_1$* through the path *movie$_1$.movie$_1$_director$_1$.director$_1$*. The expression *(director$_3$.director$_3$_movie$_3$.movie$_3$)$^{-1}$.name$_3$* specifies the attribute *name$_3$* that is related to *movie$_3$* through the relationship *director$_3$.director$_3$_movie$_3$.movie$_3$* in an reverse order.

There is one mapping view for *actor$_m$* from *actor$_1$*: *V_{actor1}({name$_1$, nationality$_1$}{});*

Given the mapping views for a mediation entity, the second step identifies candidate operators between these mapping views. The authors consider three operators between mapping views: union ($\cup_p$), intersection ($\cap_p$), difference ($-_p$). The mapping operators to be applied between two mapping views are determined based on the correspondence assertions that specify the relationships between the source entities as specified in the following table. For example, given two mapping views *V1* and *V2*, if $V1 \equiv V2$, union and intersection can be both applied to them. If $V1 \subset V2$, they can be combined as $V1 \cup_p V2$ or $V1 -_p V2$.

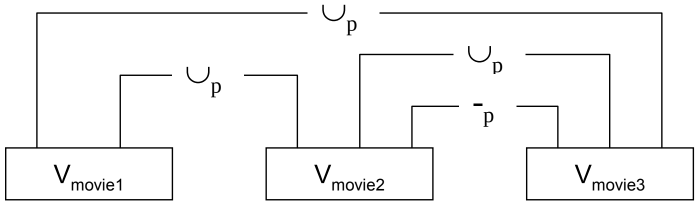


Figure 2-10. Example of an operation graph

Mediation queries are defined for every mediation entity. Mapping views for a mediation entity and the operations between them can be represented by an *operation graph*. Figure 2-10 gives an example of an operation graph for the mediation entity *movie$_m$*. Consider the operation graph. *Computation paths* are first defined such that each one is a connected subgraph of the operation graph that involves all the attributes of the mediation entity. A mediation query is a computation path with a particular order on the operations in the path. From Figure 2-10, three of the possible mediation queries are as follows:

$Q1_{movie} = (V_{movie1} \cup_p V_{movie2}) \cup_p V_{movie3}$;
$Q2_{movie} = V_{movie1} \cup_p (V_{movie2} \cup_p V_{movie3})$;
$Q3_{movie} = (V_{movie2} -_p V_{movie3}) \cup_p V_{movie1}$.

2.3. Industrial Tools

Some industrial tools are also available for automatic mapping generation such as Adeptia Integration Server [AIS], Altova MapForce [AMF] and Stylus [Sty]. These three approaches all propose the same kind of solutions: they consider one target schema and one source schema based on the XML model. They allow users to specify correspondences between elements of the schemas and to generate a mapping that can be expressed in XQuery or in XSLT.

The main objective of these approaches is to provide a friendly interface to help users to visualize target and source schemas and to specify correspondences between source elements and target elements. Users can specify 1-1 correspondences or complex 1-n correspondences to define a target element from a function over one or several source elements.

Once a user specified all the correspondences, a mapping is automatically generated from the source schema to the target schema. The mapping specifies for every target element having a correspondence how to derive its instances from the sources using the correspondence. However, generated mappings are simple mappings and they do not consider the structural constraints between different target elements. Stylus and Adeptia Integration Server derive instances for every target element without considering its location in the target schema and without considering the other elements. Altova MapForce satisfies the structural constraints between target elements only if they have a similar structural relationship in the source schema. If there are two target elements *A* and *B* such that *A* is the parent of *B* and if their respective corresponding elements in the source schema *A'* and *B'* are siblings or *A'* is a child of *B'*, Altova MapForce cannot restructure these elements. It derives instances for *A* and *B* from A' and B' respectively, but does not satisfy the structural constraint between them.

All these tools do not combine different portions of the source and they do not consider referential constraints in the source schema. If a target element is related to two correspondences, Altova MapForce duplicates the element and relates each of them to a correspondence. Stylus and Adeptia Integration Server derive the target element as the union of the two corresponding source elements without checking if one is the duplicate of the other

2.4. Discussions

Table 2-1 shows some of the main features of the existing approaches for mapping generation such as metadata used to generate mappings, characterization of the resulting mappings, and the kind of generation process.

The metadata for mapping generation gives is related to the input knowledge required by the different approaches. It includes two aspects: schema model and semantic equivalence between schema elements. Schema models can be relational, nested relational or an XML model. Equivalences between elements can be 1-1 equivalences or 1-n equivalences.

To characterize the result mappings, we consider both the semantics and the format of the resulting mappings. With respect to the semantics of the mappings, we differentiate between mappings that involve joins between different sources and mappings that do not involve joins between different sources. The mapping format is related to the formalism used to describe the resulting mappings; this can be done using a standard language or an ad-hoc one. We also distinguish between mappings that derives instances conform to the target schema and the others.

The type of mapping generation process indicates if the mapping is produced by generating directly mapping expressions or by restructuring the source schema.

Table 2-1. Main features of the different approaches on mapping generation

Approach	**Metadata for Mapping Generation**		**Characterization of Resulting Mappings**			**Type of Mapping Generation**
	Schema model	**Element Equivalences**	**Involving Inter-Source Joins?**	**Description Language**	**Mapping Conform to the Target?**	
Kedad and Bouzeghoub'99	Relational	1-1 equivalences	Yes	SQL	Yes	SQL query generation
Clio'00	Relational	1-1 , 1-n equivalences	No	SQL	Yes	SQL query generation
TUPELO	Relational	1-1 , 1-n equivalences	No	specific algebra expression	Yes	Source schema restructuring
Xtra	DTD	1-1 equivalences	No	XSLT	No	Source schema restructuring
AutoMed	DTD, XML Schema	1-1 equivalences	No	Transformation pathways	No	Source schema restructuring
ORA-SS	DTD	1-1 equivalences	No	XQuery	No	XQuery query generation
Clio'02	Nested relational	1-1 equivalences	No	Specific logical expression	No	Logical expression generation
Farias Lóscio and Salgado'03	XML Schema	1-1 equivalences	Yes	Specific algebra expressions	Yes	Algebra expression generation
Industrial tools	XML Schema	1-1 , 1-n equivalences	No	XQuery, XSLT	Yes	(no information)

Metadata for Mapping Generation

All the mapping generation approaches discussed so far consider as inputs the target schema and the source schemas. They also all assume that the correspondences between elements are given and are used to generate mappings.

In some of approaches, the correspondences between elements are specifically represented . These approaches include [BKS04, MHH00, CLL03 and FW06] and the industrial solutions. In [SKR01, Zam04, PVM02 and FS03] the correspondences between elements are implicitly represented by equivalences of element names.

All approaches assume the semantic correspondences between elements of the target schema and elements of the source schemas. The approaches proposed in [BKS04 and FS03] also consider correspondences between elements of different sources. They allow the expression of the equivalence between two source elements that do not have equivalent elements in the target schema.

Mainly two kinds of equivalences are used: 1-1 correspondences and 1-n correspondences. A 1-1 correspondence is specified between two elements to state that they represent the same concept. This is considered by all the approaches. A 1-n correspondence is used to relate a target element to one or several source elements using a transformation function. It states that instances of the target element can be derived from the result of applying the function to the instances of the source elements. 1-n correspondences allow expressing more complex relationships between elements. A 1-1 correspondence can be seen as a special case of 1-n correspondence in which the transformation function is applied on one source element and returns instances of this element. 1-n correspondences are supported by [FW06 and PVM02] and they can be specified using the industrial tools presented above.

Mapping Semantics

We distinguish between two kinds of mappings depending on if they can express joins between different sources or not. Consider the mappings containing joins between different sources allows generate more mappings conform to the target. It may happen that only these mappings can satisfy the target. Suppose that a target schema requires authors and their respective books and assume that one source *S1* describes books while the other source *S2* describes authors. If no mapping can express joins between *S1* and *S2*, books and authors can be retrieved separately for the target schema, but no mapping can satisfy the target schema by expressing how the books are related to each author. Getting this information requires joining books in *S1* with authors in *S2*.

We distinguish the existing approaches into the ones that can not generate mappings involving inter-source joins and those that can generate mappings involving inter-source joins. The first kind covers the majority of the proposed solutions including [MHH00, FW06, SKR01, Zam04, PVM02 and CLL03] and the industrial tools. They use structural constraints and referential constraints to re-organize instances of the source schema to match the target structure.

Two approaches [BKS04 and FS03] generate mappings that can join different sources. To generate these mappings, these approaches have to deal with the issues such as how to identify identical objects between different sources and how to "semantically" join them. For joining different sources, the approach proposed in [BKS04] infers joins based on element equivalences and on key and referential constraints. the approach proposed in [FS03] does not consider key constraints and uses element equivalences to infer joins.

The approaches proposed in [MHH00 and Zam04] consider using the generated mapping for single source schema to obtain mappings from several sources. The generation consists in two steps: mappings are first generated for every source using a restructuring mapping generation approach, and then the mappings for different sources are combined using a union to get a mapping involving multiple sources. However, these methods are still different from generating integrating mappings. If we consider the example presented at the beginning of the section, using union between authors and books do not allow finding relationships between every book and its authors.

Mapping Description Language

Some approaches generate mappings in a standard language with respect to its data model: the approaches proposed in [MHH00, BKS04] generate SQL queries; the approach proposed in [CLL03] generates XQuery queries; Xtra [SKR01] generates XSLT scripts; and all the three industrial solutions generate both XQuery queries and XSLT scripts. Other approaches [FS03, FW06, PVM02 and Zam04], generate mappings in a specific high-level format.

Mappings expressed in a standard language can be directly used in the systems supporting the language. Mappings expressed in an abstract language will require the translation of mappings to be used in these systems.

The mappings generated by [PVM02] are expressed in an abstract language and an algorithm [YP04] is proposed to rewrite user queries for this language. Mappings generated by [Zam04] are also expressed in an abstract language and they are mainly used within the AutoMed project [BKL04].

Conformance of the Mappings to the Target Schema

Depending on the relationship between the resulting mappings and the target schema, we differentiate the existing approaches into two kinds: those generating mappings that always derive instances conforming to the target structure, and the others. We illustrate the difference between these two kinds of approaches by an example. Consider a source that contains information about some authors and their books and a target schema that specifies books with their authors and their publisher. The first kind of approaches can not generate mappings in this case because the source do not contains information about books' publisher required by the target. The second kind of approaches can generate a mapping to re-organize the instances of the source to the target structure: classify the authors by their books. This mapping does not derive instances conforming to the target structure because the books' publishers are not given by the source. But the structure of the instances is homogeneous with respect to the target one.

We can consider that those approaches that do not have to generate mappings deriving instances conform to the target structure mainly focus on restructuring the source instances to a structure that is homogeneous with the target structure. This kind includes the approaches [SKR01, Zam04, CLL03 and PVM02]. Among these approaches, [Zam04 and CLL03] are both in a system where the target schema is generated as the integration of the source schemas and the system considers a Local-As-View approach. Therefore mappings only need to define the restructuring of the instances of every source to a structure homogeneous with respect to the target one. [SKR01 and PVM02] offer no discussion for this subject. They generate a mapping without checking if the generated mappings derive instances conform to the target.

The other kind of approaches mainly focuses on defining the target schema over the sources. It includes [BKS04, MHH00, PVM02, FW06, FS03] and all the industrial approaches. To compare with the first kind, their generated mappings derive instances conform to the target. However, no mapping can be generated for the target schema even only a fraction of the target cannot be found in the sources.

Number of Result Mappings

Most of the approaches generate only one single mapping result, such as [MHH00, FW06, SKR01, Zam04, CLL03 and PVM02] and the industrial solutions. If several possibilities exist for deriving instances from the source to the target, one of them is chosen based on a cost model [SKR01] or based on some heuristics [FW06]. There are two approaches [BKS04 and FS03] that generate a set of alternative mappings. In this set, each mapping represents a different way to derive instances from the sources to the target independent from the other mappings.

The approach proposed in [PVM02] generates one single result that is a set of mappings: each one defines a portion of the target schema from a portion of the source. Two target portions respectively defined by two different mappings may be overlapping.

The advantage of the approaches generating several mappings is that they provide users a choice to select the most relevant one, that is, the one having the semantics that corresponds to their needs.

In different use cases, a user may prefer one or the other. It may also be possible that in a given context, two users prefer different mappings. However, providing alternative mappings also complicates the mapping generation process since all possible ways to relate the target to the source need to be enumerated. Mappings generated by [PVM02] can be considered as a hybrid solution of the two previous ones. But mappings are specified in an ad-hoc format.

Schema Restructuring versus Mapping Expression Definition

With respect to the method used to generate mappings, we distinguish between approaches that restructure source schemas and the approaches that generate mapping expression definitions without restructuring.

The former generate mappings by restructuring the source schema to the target. This includes TUPELO, Xtra and AutoMed. These approaches define a set of restructuring operators. Generating mappings consists in producing a sequence of restructuring operators to restructure the source schema to the target structure. Sometimes sequence operations are later translated to a more declarative query between the source schema and the target schema. There are some approaches [CR03 and THH05] that do not generate mappings but they propose a set transformation operators that can be used to specify mappings.

The second kind of approaches generates directly declarative expressions to define the target schema based on the sources. This includes Kedad and Bouzeghoub'99, Clio'00, ORA-SS, Clio'02, as well as Farias Lóscio and Salgado'03.

2.5. Limitations of the Existing Approaches

Most of the existing approaches generate mappings from one source schema. The approaches proposed in [BKS04 and FS03] generate mappings that can involve joins between different sources. The approach proposed in [BKS04] consider relational schemas and the approach proposed in [FS03] consider XML schemas; they assume that the objects are expressed using the same constructs in the source schemas as in the target schema. A target `complexType` element can only be derived from an equivalent `complexType` element in the sources. This represents a limitation because `complexType` type in XML Schema only corresponds to a grammar definition and the same data can be specified as `complexType` elements or as `simpleType` elements with attributes. We can see that in the example of Figure 2-9, no mapping can be generated for $actor_m$ using S_2 even if there is an attribute $actor_2$ in S_2 and $actor_2$ is equivalent with $name_m$ of $actor_m$. Moreover, the approach does not consider key and referential constraints. The generated queries are expressed in an ad-hoc language.

Among the approaches that generate mappings without joins between sources, Xtra and ORA-SS also assume that the source schema and the target schema have homogeneous structures. TUPELO, AutoMed and Clio'02 generate mappings expressed in an abstract language. Clio'02 generates a set of mappings such that each one is for a portion of the target schema. Two mappings for the same schemas populate different target portions and they may be overlapping. With this result, the system cannot produce a mapping or a set of alternative mappings for the target from the sources and it cannot know either if the target is satisfied by the sources. TUPELO requires users to input example instances for the target schema and for the source to perform the mapping generation. Users need to fully understand all the schemas and their relationships for editing these examples instances. The existing industrial tools are very useful for specifying correspondences between elements. But the generated mappings are still "simple" in a way that they cannot express integration of different portions of sources and they cannot always express the restructuring from the source to the target such as restructuring elements in tree structure, etc.

2.6. Our Proposal

We propose an approach to generate mappings between one target schema and some source schemas; both target and source schemas are described through XML Schema [xsd]. It considers a set of 1-1 or 1-n semantic correspondences between the target schema and the source schemas. The result of the generation process is a set of mappings: every one represents an alternative way to define the target from the sources independently from the other result mappings.

Our basic idea is to generate mappings in three steps. To handle the complexity of mapping generation for the whole target schema, the target schema is first decomposed into subtrees; each subtree is such that, except for the root, all the other elements in the subtree are mono-valued.

Given the set of all the subtrees of the target schema, mappings are defined for each subtree to derive its instances from the instances of the source schemas. In a subtree, since except the root, all the other elements are monovalued, every instance derived by a mapping for the subtree have to contain a value for each element, no matter how the values of different elements are organized. We can therefore generate mappings for a subtree without looking for the hierarchical relationships between its elements. This process is inspired by [BKS04]. The result mappings for subtrees can involve joins between different schemas.

Partial mappings of the different subtrees in the target subtree are combined to generate the mappings for the whole schema. Every combination that satisfies the parent-child relationships between the subtrees lead to a target mapping. Mappings are expressed in an abstract language and they can be translated into another language. We propose an algorithm to translate the abstract queries into XQuery [xquery].

If the data sources do not allow the generation of a mapping that satisfies users' needs as represented by the target schema, our approach proposes to relax some cardinality constraints or structural constraints in the target schema such that a mapping can be found for the new schema. This process is interactive with respect to users who have to review the proposed schemas and have to decide between alternative solutions.

To compare to the existing approaches discussed so far, our approach considers target and source schemas expressed in XML Schema. It assumes having 1-1 and 1-n correspondences between the target elements and the source elements. It generates a set of mappings to specify different ways of defining the target and they can involve joins between different sources that are inferred using the semantic correspondences and constraints; it considers relative keys that are valid only inside of a portion of the schema (e.g. the chapter numbers can be considered to be a key for chapters, but the key is only valid inside of the books).

The resulting mappings are specified in an abstract language and they can be translated into XQuery. The approach generates mappings that conform to the target schema. If no mapping can be generated to satisfy the target, it proposes to relax some constraints of the target schema to enable generating mappings satisfying the new schema.

Our approach does not assume that the objects are described using the same constructs in the target and the source schemas. Consider the example shown above in Section 2.5 that does not allow generating a mapping using Farias Lóscio and Salgado'03, the same example allows generating a mapping using our approach. It does not require users to provide these critical instances as required by TUPELO.

3. Mapping Adaptation

Mappings are expressions defining a target schema from some source schemas and they necessarily depend on the schemas to which they relate. However, in a distributed environment, sources are autonomous and freely change their contents, as well as the target schemas may also evolve for integrating new needs. When one of these schemas evolves, the mapping may become obsolete and need to be redefined. Automatic mapping adaptation consists in making use of the original mapping and automatically adapting it to the new schemas.

Several solutions have been proposed for automatic mapping adaptation: EVE, Bouzeghoub et al.'03, Farias Lóscio and Salgado'04, ToMAS, AutoMed and MACES. They can be distinguished into two kinds: ***incremental approaches*** and ***mapping composition approaches***. Incremental approaches consist in incrementally adapting the mapping by applying isolated modifications for every change occurring in the system. Mapping composition approaches consider a mapping describing the evolution and the adaptation is done by compositing such mapping with the original one.

In the remainder of this section, we first present some the incremental approaches in Section 3.1. Mapping composition approaches are described in Section 3.2. A discussion on these approaches is given in Section 3.3. Section 3.4 presents some the limitations of the existing approaches and Section 3.5 presents our proposal and compares it to the related works.

3.1. Incremental Approaches

The main idea of the incremental approaches is that schemas often evolve in small, primitive steps; after each of these steps, the mapping can be incrementally adapted by applying isolated modifications. The schema changes are represented using a sequence of elementary change operations (e.g., adding an element, removing an element, removing a constraint etc.). The mapping is then adapted following a predefined adaptation strategy for each of these elementary operations. The approaches belonging to this category focus mainly on (i) defining a list of elementary change types, and (ii) specifying a mapping adaptation strategy for each elementary change type.

The incremental approach, designed to handle elementary changes, is intuitive and efficient for the cases where there is not a drastic evolution from the original schema to the changed schema. It also allows a system to process specific adaptation actions for each change. For example, the system may search substitutions for element removal and make some syntactical changes for element renaming. However, there exist also some limits. The algorithm is applied after each elementary change. This can become inefficient in a context where changes occur frequently. Consider one change that is followed by its reverse change (e.g. adding an element in a source and then removing it), the mapping adaptation will be done consequently for each of them.

Four approaches have been proposed for incremental mapping adaptation: EVE, Bouzeghoub et al.'03, Farias Lóscio and Salgado'04 and ToMAS. We present the principles of these approaches in the rest of the section.

EVE (Evolvable View Environment)

EVE has been proposed in the context of data warehousing where the mapping is called view and the mapping adaptation is called view synchronization. Mappings are defined over several relational sources and the views are defined in SQL. The main contributions of this approach are: (i) defining an extension of SQL (E-SQL) for defining views with evolution preferences; (ii) defining what constitutes a legal view rewriting (mapping adaptation) using source descriptions.

E-SQL is an extension of `SELECT-FROM-WHERE` queries augmented with specifications of how the query may be evolved under source schema changes. Figure 2-11 shows two views defined using E-SQL. The attributes (A) in the `SELECT` clause, relations (R) in the `FROM` clause, and primitive clauses (C) in the `WHERE` clause are the basic units of a view, called *view components*. Two evolution parameters are attached to each view component. The *dispensable parameter* states whether the view component is required and, hence, must be kept in the evolved view (when the value is false). The dispensable parameter is denoted *XD*, where *X* is *A*, *R*, or *C* for attribute, relation, or elementary clause component respectively. The *replaceable parameter* specifies whether the view component can be replaced in the view synchronization process (when the value is true). It is denoted *XR* where *X* is defined as above. The default value of the parameter is false.

With E-SQL, a view can also specify the *view extent parameter*: if the evolved view extent must be equivalent to ("≡"), a superset of ("⊇"), or a subset of ("⊆"), with respect to the original view extent. This is defined using the *VE* parameter. *VE* is set to "≈" if no restriction is given. The default value of *VE* is "≡".

Views are evolved using the provided description of the sources. A common model is used to describe source descriptions in which we have following components:

- data content description associated to the data in the sources. A relation *R* of the information source (*IS*) and the set of attributes belonging to *R* are described as follows: $IS.R(A_1, \ldots, A_n)$.
- type integrity constraints describing the domain types of attributes.
- join constraints between relations. Every join constraint is between two relations R_1 and R_2 to state that tuples in R_1 and R_2 can be meaningfully joined if the join condition is satisfied. It is denoted by $JC_{R1, R2} = (C_1 \text{ AND } \ldots \text{ AND } C_l)$, where $C_1, \ldots, C_l$ are elementary clauses over the attributes of R_1 and R_2.
- partial/complete information constraint between relations. Every partial/complete constraint is between two relations R_1 and R_2 to state that a fragment of R_1 is semantically contained or equivalent to a fragment of R_2 at all times. Its denoted by $\pi_{Ai1, .., Aik}(\sigma_{C(Aj1, .., Ajt)}R_1)$ $\theta\ \pi_{An1, .., Ank}(\sigma_{C(Am1, .., Amt)}R_2)$, where $Ai_1, .., Ai_k$ and $Aj_1, .., Aj_t$ are attributes of R_1 and $An_1, .., An_k$ and $Am_1, .., Am_t$ are attributes of R_2; and $\theta = \{\subseteq, \supseteq, \equiv\}$ for the partial ($\subseteq$ and $\supseteq$) or complete ($\equiv$) information constraint, respectively. For example, $\pi_{Name, Address}(Person) \supseteq \pi_{Name, Address}(Customer)$ states that the *Customer* relation is contained in the *Person* relation.

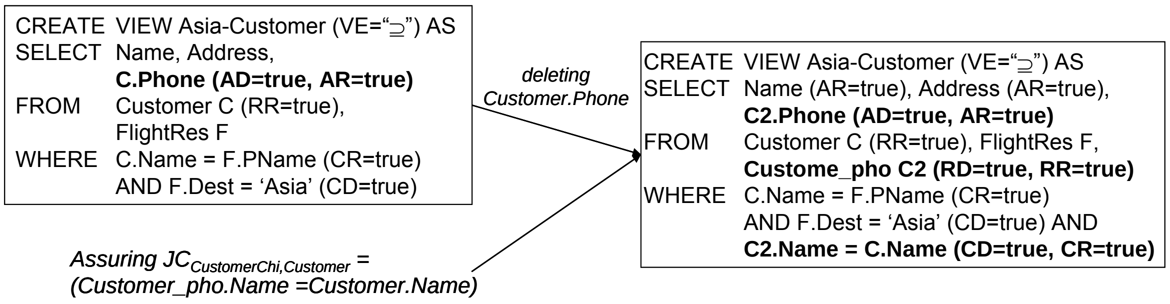
The EVE approach considers (i) adding, removing and renaming relations and (ii) adding, removing, and renaming attributes. For every change, a view becomes *affected* if any of its view components is affected by the change. It is *amendable* if none of the affected view components has its evolution parameters set to (false, false).

Only affected views need to be evolved. Adding attributes and relations does not affect the existing views and therefore does not trigger view synchronization. Attribute and relation renaming may affect the view but it just requires syntactic changes. Removing attributes and relations may also make the view *affected* which is described in the following.

When an attribute *R.A* referred to in the view *V* (in the *SELECT* or *WHERE* clauses) is deleted from its site, the view synchronizer attempts to find a substitute to replace *R.A* if it is replaceable. An attribute *S.B* is said to be an appropriate substitute for *R.A* if:

- *S.B* has the same domain type as *R.A*.
- There is a meaningful join relationship between the relations R and S.
- Based on the value δ of the view extent parameter of *V* and the join condition *C* defined by the previous point, $\pi_{((Attr(V) \cap Attr(R)) \setminus \{R.A\}) \cup \{S.B\}}(R \bowtie_C S)\ \delta\ \pi_{((Attr(V) \cap Attr(R)) \setminus \{R.A\}) \cup \{R.A\}}(R)$ must hold. $(Attr(V) \cap Attr(R)) \setminus \{R.A\}$ denotes all attributes of R that are in view except *A*.

Consider that the attribute *Customer.Phone* is deleted from the view at the left side of Figure 2-11. The view at the right side gives an example of the evolved view. Assuming another relation *Customer_pho* having an attribute *Phone* such that *Customer_pho.Phone* and *Customer.Phone* have the same domain type; *Customer_pho* can be joined with *Customer*; and the view extent parameter is satisfied, the view synchronizer of EVE substitutes *Customer.Phone* with *Customer_pho.Phone*.



```
CREATE  VIEW Asia-Customer (VE="⊇") AS
SELECT  Name, Address,
        C.Phone (AD=true, AR=true)
FROM    Customer C (RR=true),
        FlightRes F
WHERE   C.Name = F.PName (CR=true)
        AND F.Dest = 'Asia' (CD=true)
```

```
CREATE  VIEW Asia-Customer (VE="⊇") AS
SELECT  Name (AR=true), Address (AR=true),
        C2.Phone (AD=true, AR=true)
FROM    Customer C (RR=true), FlightRes F,
        Custome_pho C2 (RD=true, RR=true)
WHERE   C.Name = F.PName (CR=true)
        AND F.Dest = 'Asia' (CD=true) AND
        C2.Name = C.Name (CD=true, CR=true)
```

deleting Customer.Phone
Assuring $JC_{CustomerChi,Customer}$ = *(Customer_pho.Name =Customer.Name)*

Figure 2-11. Deleting attributes

When a relation *R* in the *FROM* clause of a view *V* is deleted, another relation needs to be found to substitute *R*. A relation *S* is said to be an appropriate substitute for *R* if the following three conditions are satisfied:

- The relation *S* must contain the corresponding attributes of the relation *R* that are indispensable and replaceable in the view *V*.
- Every attribute of *S* that is used as replacement for an attribute of *R* must have the same domain type.
- Assuming the value of the view extent parameter of *V* being δ, the following condition must hold: $\pi_B(S)\ \delta\ \pi_A(R)$, where *B* are the attributes of *S* that are used as replacements for the attributes *A* of *R*.

Several solutions may be found to substitute affected view components, and any one of them can be chosen.

When a view component *C'* is used to replace an affected view component *C*, the values of the evolution parameters for *C'* are assigned using the following rules:

- If *C'* is used to replace exactly one view component *C*, the new evolution parameters are set to be the same as those of *C*.
- If *C'* is used to replace several view components $X_1, \dots, X_k$ with evolution parameter settings being $X_i(par_{i,1} = val_{i,1},\ par_{i,2} = val_{i,2})$, where $par_{i,1}$ and $par_{i,2}$ are two evolution parameters and $val_{i,j} \in \{true, false\}$. Then the two evolution parameters of C' are:

  $par_{C,1} = val_{1,1}\ AND \dots\ AND\ val_{k,1}$;

  $par_{C,2} = val_{1,2}\ AND \dots\ AND\ val_{k,2}$.

Bouzeghoub et al.'03 [BFK03]

This work is the continuation of Kedad and Bouzeghoub'99 [KB'99 and BKS'04], which is presented in Section 2.2 to propose a design methodology to generate mediation queries based on the relational model. The work of Bouzeghoub et al.'03 can be seen as an incremental execution of this algorithm. It assumes that the mediation query generation system maintains all intermediate results of the generation process that are operation graphs (relevant relations and candidate operations) and the computation paths. For every change in the source, the incremental algorithm updates step-by-step the affected parts of each intermediate result.

Source changes are propagated to the mediation queries. The following changes are considered (i) adding or removing a source attribute, (ii) adding or removing a source referential constraint and (iii) adding or removing a source relation.

A list of propagation primitives is also defined to specify elementary changes that can be applied to the operation graph independently from any particular source change. Among them are the following primitive:

- *valid_operation_graph(G_{Rm})*: checks the validity of the operation graph G_{Rm} associated with the relation Rm and removes the invalid set based operations;
- *search_operation(G_{Rm})*: searches for new operations for combining pairs of relevant relations in the operation graph G_{Rm};
- *add_relevant_relation(Ti, G_{Rm})*: adds the relevant relation Ti into the operation graph G_{Rm};
- *remove_relevant_relation(Ti, G_{Rm})*: removes the relevant relation Ti and the operation involving Ti from the operation graph G_{Rm}
- *search_relevant_relation(G_{Rm}, S)*: searches a new relevant relation associated with relation Rm from the set of data sources S.
- *generate_query(G_{Rm}, Q)*: generates the set Q of relational expressions to compute the relation Rm using the operation graph G_{Rm};

Given a set of changes considered by the system and a set of propagation primitives, evolution rules are defined to relate every change with a set of propagation primitives. Every evolution rule is an event-condition-action (ECA) rule in which the event is a change and the action is a set of propagation primitives to execute when the conditions are satisfied. Figure 2-12 shows three evolution rules. Rule 1 and 2 propagate attribute additions in the mediation queries. If the added attribute belongs to a source relation corresponding to a relevant relation of the operation graph, rule 1 (i) first adds the new attribute to the actual relevant relation *Ti*, (ii) then checks for removing set-based operations involving *Ti* such as union and difference; (iii) finally searches for new operations to relate *Ti*. If the added attribute does not belong to a source relation corresponding to a relevant relation in the operation graph, rule 2 (i) adds a relevant relation in the operation graph that concerns the added attribute and the primary key and foreign keys in the same source relation; (ii) and searches for new operations to relate the new relevant relation. Rule 3 defines the evolution actions for the removal of a source relation *Si*. If *Si* corresponds to a relevant relation *Ti* in the operational graph, the rule removes *Ti* from the operation graph with all the operations that involve *Ti*.

Rule 1 (Rm) **Event**: add_attribute(Si, A) **Condition**: A $\in$ Rm $\exists$ (Ti $\in$ M_{Rm} | Ti $\subseteq$ Si) /*M_{Rm} is the set of relevant relations for Rm*/ **Action**: Ti := Ti $\cup$ {A}, valid_operation_graph(GRm), search_operation(G_{Rm})	**Rule 2 (Rm)** **Event**: add_attribute(Si, A) **Condition**: A $\in$ Rm $\neg\exists$ (Ti $\in$ M_{Rm} | Ti $\subseteq$ Si) **Action**: /* X is the set of key attributes and foreign keys of Si */ Ti := $\Pi_{X \cup A}$ Si, add_relevant_relation(Ti, GRm), search_operation(G_{Rm})	**Rule 3 (Rm)** **Event**: remove_relation(Si) **Condition**: $\exists$ (Ti $\in$ M_{Rm} | Ti $\subseteq$ Si) **Action**: remove_relevant_relation(Ti, GRm)

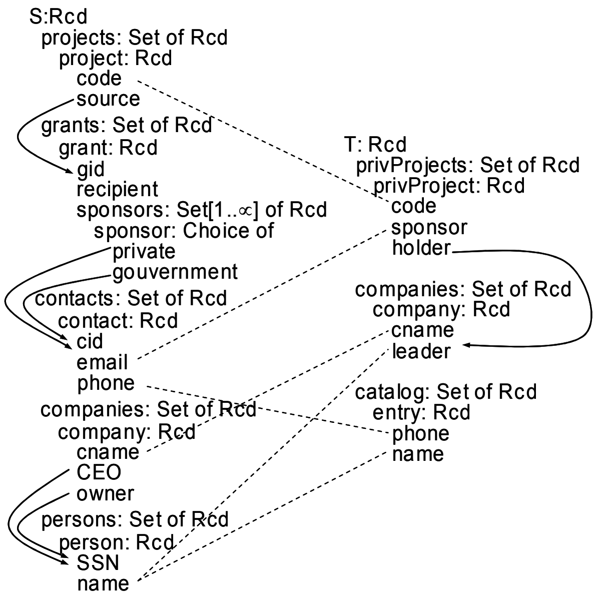
Figure 2-12. Three evolution rules

Once the operation graph is reorganized for the source changes, mediation queries can be generated from the new operation graph if its corresponding relevant sources have been modified.

Farias Lóscio and Salgado'04 [FS04] makes use of this approach to evolve mappings generated by Farias Lóscio and Salgado'03 [FS03]. The latter generates mappings for every entity using the process of [BKS04] that generates mappings for a target relation. [FS04] therefore evolves mappings for each entity in the same way as Bouzeghoub and al.'03 evolves mappings for each target relation.

ToMAS (Toronto Mapping Adaptation System) [VMP03]

ToMAS is a tool for automatically adapting mappings generated by Clio'02 [PVM02] that generates a set of mappings between a target schema and a source schema.



```
m1:
foreach   S.projects p, S.grants g, S.contacts c,
          g.grant.sponsors n, n.sponsor.private r
  where   p.project.source=g.grant.gid and
          r=c.contact.cid
exists    T.privProjects j, T.companies m
  where   j.privProject.holder=m.company.cname
with      j.privProject.code=p.project.code and
          j.privProject.sponsor=c.contact.email
m2:
foreach   S.companies c, S.persons p
  where   p.person.SSN=c.company.owner
exists    T.companies o
with      o.company.cname=c.company.cname
          and o.company.leader=p.person.name
m3:
foreach   S.contacts c, S.persons p
  where   p.person.SSN=c.contact.cid
exists    T.catalog e
with      e.entry.name=p.person.name and
          e.entry.phone=c.contact.phone
```

Figure 2-13. Two schemas and three mappings between them

```
A1: select  *
from S.projects p, S.grants g, S.contacts c,
       g.grant.sponsors n, n.sponsor.private r
where p.project.source=g.grant.gid and
       r=c.contact.cid

A2: select  *
from  S.projects p, S.grants g,
       g.grant.sponsors n,
       n.sponsor.gouvernment r, S.contacts c
where p.project.source=g.grant.gid and
       r=c.contact.cid

A3: select  *
from  S.grants g, S.contacts c,
       g.grant.sponsors n, n.sponsor.private r
where r=c.contact.cid

A4: select  *
from  S.grants g, S.contacts c,
       g.grant.sponsors n,
       n.sponsor.gouvernment r
where r=c.contact.cid
```

```
A5: select  *  from S.contacts c

A6: select  *
from S.companies c, S.persons p,
       S.persons w
where c.company.CEO=p.person.SSN
     and c.company.owner=w.person.SSN

A7: select  *  from S.persons c

A8: select  *
from S.contacts c, S.persons p
where   c.person.SSN=p.contact.cid

B1: select  *
from T.privProjects j, T.companies m
where j.privProject.holder
           =m.company.cname

B2: select  *   from T.companies m

B3: select  *   from  T.catalog e
```

(a) Logical associations for the schemas S and T

```
A1a:
select *
from     S.projects p, S.grants g, S.contacts c,
         g.sponsors.sponsor.private r,
         S.companies o, S.persons e, S.person e'
where    p.project.source=g.grant.gid and
         r=c.contact.cid and
         g.grant.recipient=o.company.cname and
         o.company.CEO=e.person.SSN and
         o.company.owner=e'.person.SSN
m1a:
foreach S.projects p, S.grants g, S.contacts c,
         g.sponsors.sponsor.private r,
         S.companies o, S.persons e
where    p.project.source=g.grant.gid and
         r=c.contact.cid and
         g.grant.recipient=o.company.cname and
         o.company.CEO=e.person.SSN
exists   T.privProjects j, T.companies m
where    j.privProject.holder=m.company.cname
with     j.privProject.code=p.project.code and
         j.privProject.sponsor=c.contact.email and
         m.company.cname=o.company.cname
         and m.company.leader=e.person.name
```

(b) New logical association and mapping after the mapping adaptation for a constraint removing

Figure 2-14. Logical associations for S and T and new mapping after a constraint removing

Figure 2-13 shows one target schema, one source schema and three mappings for them. Let *S* and *T* be a pair of source and target schemas, each mapping is a *foreach A^S exists A^T with D* expression such that A^S and A^T are logical associations in *S* and *T* and *D* contains the conditions of the correspondences in *C* that concern elements in the pair $<A^S, A^T>$. Figure 2-14 (a) shows the logical associations for the schemas of Figure 2-13: *A1* to *A8* for *S* and *B1* to *B3* for *T*. Logical associations

A1 to *A7* and *B1* to *B3* are automatically generated following the above rule. *A8* is additionally specified by the user.

The mapping adaptation is done for each affected mapping in the system. ToMAS considers the following changes such as: (i) adding and removing referential constraints; (ii) adding and removing elements; and (iii) renaming, moving or copying elements. These changes may be in the source schema or in the target schema. Since mappings consist in pairs of logical expressions from the target and the source, the principle of mapping adaptation is similar for source evolution and for target evolution. We therefore only present their mapping adaptation algorithm for the source evolutions. Among them, renaming a schema element is mainly a syntactic change.

If a referential constraint *F* is added, the mapping *m*: *foreach* A^S *exists* A^T *with D* needs to be adapted if *F* originated from an element that can be reached by A^S. In this case, A^S is chased [PT99] with the set of the old schema constraints enhanced with the new constraint F. The chase enumerates logical join relations based on these schema constraints. The result is a set of new logical associations and a new mapping is generated for each one. These new mappings are added in the set of result mappings and for each mapping *m'* in *M* included by one of these new mappings, *m'* is removed. Assume a new constraint *S.grants.grant.recipient* referencing *S.companies.company.cname* is added to the source schema, then this change affects *m1*. The logical association for *S* of *m1* is chased using the new constraint and gives a new logical association *A1a*, that is shown in Figure 2-14 (b). This association generates two adaptations for m1 depending on if the leader element of the target is mapped as the name of CEO or the name of the owner of the company. The first one *m1a* is shown in Figure 2-14 (b), while the second *m1b* is the same as *m1a* except that *c.company.CEO* is replaced by *o.company.owner*.

If several mappings are generated from one logical association, one is chosen based on its similarity with the old mapping. Among the two adaptations *m1a* and *m1b* for *m1*, *m1a* is more similar to *m1* than *m1b* because both *m1* and *m1a* populate *c.company.CEO* compared to *m1b* which populates *o.company.owner*. Then *m1a* is kept and *m1b* is discarded.

Once a constraint *F* is removed, the mapping covering the constraint needs to be adapted. The approach isolates the affected logical association, and re-chases them by considering the set of schema constraints without *F*. If we remove the constraint *S.grants.grant.recipient* referencing *S.companies.company.cname*, the logical association of *S* in *m1a* is broken into two parts and lead to two mappings *m1* and *m2*.

Adding new structure does not lead to new mappings because ToMAS does not consider the addition of new correspondences. When an atomic element e is removed, each constraint F using e is removed and mappings are adapted. If the atomic element e is used in a correspondence V, then every mapping *m* that is covering V has to be adapted by removing the correspondence from the mapping. Removing *code* will invalidate mapping *m1*, which will be adapted by removing the equality *i.privProject.code=p.project.code* from the *with* clause.

If a schema element *e* is moved from a location *n1.n2….e* to a new location *m1.m2….e*, existing constraints that use *e* also become invalid. New constraints have to be defined. For every constraint using *n1.n2….e*, one new constraint is defined by using *m1.m2….e* instead of using *n1.n2….e*. Therefore, moving elements causes (i) removing constraints/elements and (ii) adding constraints/elements. If the element is copied instead of being moved, the same reasoning is used except that the original mappings and constraints are not removed from the mapping system as in the case of a move.

3.2. Mapping Composition Approaches

Mapping composition is the other kind of mapping adaptation approaches. Unlike from the incremental approaches, mapping composition approaches consider that changes of a schema are expressed by a mapping, called ***evolution mapping.*** The adaptation of an original mapping is therefore done by composing it with the evolution mapping. The approaches belonging to this type focus mainly on (i) specifying the evolution mapping and (ii) defining the algorithm of composing the original schema with the evolution mapping.

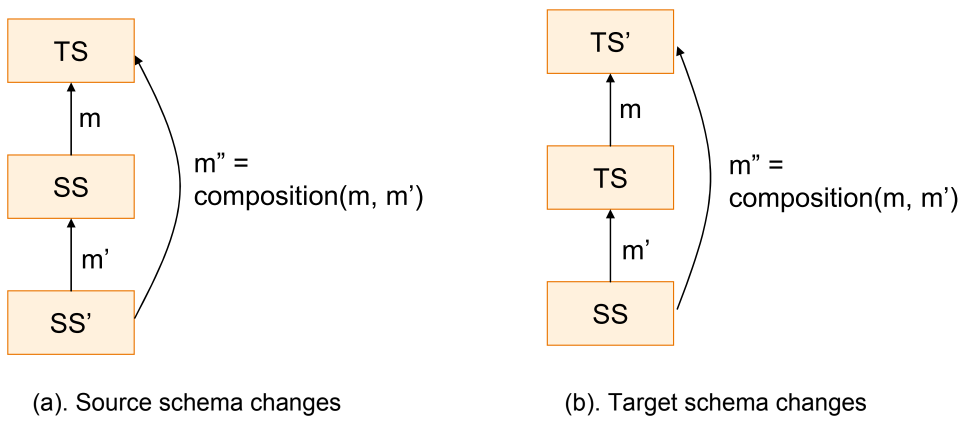


Figure 2-15. Target schema and source schema evolution in mapping composition approach

Figure 2-15 illustrates the principles of mapping composition approaches. There are two possible scenarios: one for the source schema changes and one for the target schema changes. The first one is shown in Figure 2-15 (a). Consider the target schema *TS*, the source schema *SS*, and the mapping between them *m*. If *SS* is changed to a new schema *SS'* and the evolution mapping *m'* is specified to define *SS* from *SS'*, the new mapping *m"* between *TS* and *SS'* is generated by composing *m* and *m'*. The scenario of target schema changes is illustrated in Figure 2-15 (b). Consider that the mapping between the target schema *TS* and the source schema *SS* is *m'*. If *TS* is changed to a new one *TS'* and the evolution mapping *m* is specified to define *TS'* from *TS*, the new mapping *m"* between *TS'* and *SS* is generated by composing *m* and *m'*.

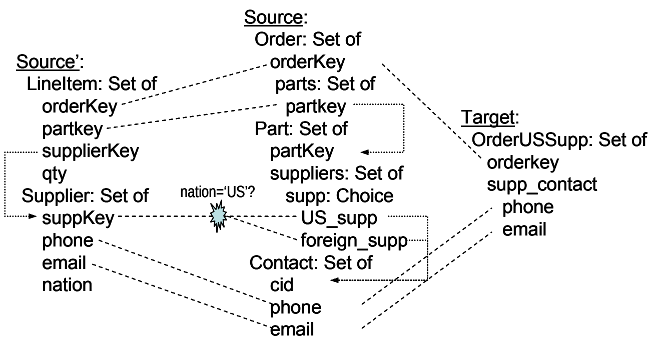
Using mapping composition approaches allows the adaptation once for a set of changes. Moreover, the system does not need to distinguish between the changes in the target and the changes in the source schemas. As shown in Figure 2-15, either the target schema or the source schema changes, the adaptation consists in composing two mappings *m* and *m'* to obtain the mapping *m"*.

The mapping composition approaches have also its limitations. The definition of the evolution mapping is a complex process. If the schema that has evolved is very complex, generating the evolution mapping may be more costly than generating mappings directly for the new schemas. Mappings are less intuitive than schema changes to express the difference between the original schema and the new schema. It is therefore difficult to analyze schema changes and optimize the adaptation process. Using this approach in a context only having few changes makes a simple problem more complex.

Two adaptation approaches by mapping composition have been proposed: AutoMed and MACES. The principle of these approaches is presented in the following.

MACES (Mapping Adaptation, using Composition, for Evolving Schemas) [YP05]

Like ToMAS, MACES is another tool for automatically adapting mappings generated by Clio'02 [PVM02], which generates a set of mapping between a target schema and a source schema. Unlike ToMAS which is an incremental approach, MACES is a mapping composition approach.



```
m:
foreach    o in Order, p in o.parts, p' in Part,
           s in p'.suppliers, i in case s.supp->US_supp,
           c in Contact
  where    p.partKey=p'.suppliers and i=c.cid
exists     os in OrderUSSupp
with       os.orderKey=o.orderKey and
           os.supp_contact.phone=c.phone and
           os.supp_contact.email=c.email

e1:
foreach    l in LineIte, $s_0$ in Supplier
  where    l.suppKey=$s_0$.suppKey and $s_0$.nation='US'
exists     o in Order, p in o.parts, p' in Part,
           s in p'.suppliers, i in case s.supo->US_supp,
           c in Contact
  where    p.partKey=p'partKey and i=c.cid
with       o.orderKey=l.orderKey and
           p.partKey=l.partKey and i=$s_0$.suppKey and
           c.phone=$s_0$.phone and c.email=$s_0$.email

e2: similar to e1, but with $s_0$.nation≠'US' as source filter, and i in
case s.supp->foreign_supp as target choice selection
```

Figure 2-16. A mapping adaptation scenario in MACES

Consider three schemas S_1, S_2 and S_3, a mapping m_{12} between S_1 and S_2 and another m_{23} between S_2 and S_3. MACES composes m_{12} and m_{23} to get the possible mappings between S_1 and S_3. The result may contain several mappings and it is denoted M_{13}.

Composing two mappings m_{12} and m_{23} consists in three steps. The mapping m_{12} is firstly skolemized into a set of rules to express how the elements of S_2 are expressed using elements of S_1. In each rule, the elements of S_2 that are not directly related to a source term in the *with* clause of m_{12} are assigned by a function *F(t1, …, tk)*, where *t1, …, tk* are the source terms that appear in the *with* clause of m_{12}. These rules are used to modify m_{23} by translating all references to S_2 into references to S_1. The result of the modification is M_{13}. Finally the mappings in M_{13} are checked to see if they are valid, which means if the terms of S_3 are all related to terms of S_1.

Figure 2-16 depicts the mapping *m* between *Source* and *Target* and the two evolution mappings *e1* and *e2* between *Source'* and *Source*. With respect to *e1*, the following rules for *Part* and *Part/suppliers* for *Source* are created:

```
Part =         for $l_0$ in LineItem, $s_0$ in Supplier
               where $l_0$.suppKey=$s_0$.suppKey and $s_0$.nation = 'US'
               return [partKey=$l_0$.partKey, suppliers=SKs($l_0$.partKey)]
SKs(p) = for $l_1$ in LineItem, $s_1$ in Supplier
               where $l_1$.suppKey=$s_1$.suppKey and $s_1$.nation='US' and p=$l_1$.partKey
               return [supp=<US_supp=$s_1$.suppKey>]
```

The rules are used to replace source terms in the mapping *m* in Figure 2-16 and the result is reduced such as all equalities between function terms are replaced by the equalities of their arguments. Only one mapping is obtained and it is as follows:

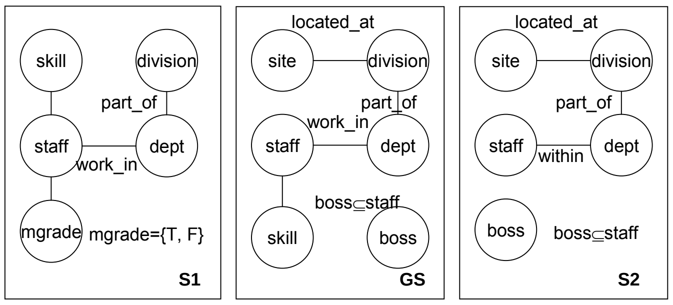
```
foreach     l in LineItem, l' in LineItem, $s_0$ in Supplier
  where     l.partKey = l'.partKey and l'.suppKey = $s_0$.suppKey and $s_0$.nation='US'
exists      os in OrderUSSupp
 with       os.orderKey=l.orderKey and os.supp_contact.phone=$s_0$.phone
            and os.supp_contact.email=$s_0$.email
```

The composition of *e2* and *m* is dropped during the reduction phase because the term *US_supp* in *m* does not have matching functions with *e2*.

There is a set of mappings between two schemas. Consider *N* mappings between S_1 and S_2 and *M* mappings between S_2 and S_3, then there are *M*N* combinations to get the mappings between S_1 and S_3. For reducing the number of combinations, MACES presents a method that filters the unaffected original mappings, and the evolution mappings that do not need to participate in the adaptation. The mapping pruning is done in three steps. It first removes all the original mappings that are not affected. Then it searches pairs of the original mapping and the evolution mapping such that these two mappings reference common parts of S_2. For every mapping *m* between S_1 and S_2, it looks for mappings between S_2 and S_3 in which sources terms can be expressed by *m*. At the end, it removes redundant mappings by using containment relationships between evolution mappings. For every two evolution mappings *e1* and *e2*, if *e1* contains *e2*, then all combinations containing *e2* are removed.

AutoMed [MP02, FP04]

This work is done in the context of the AutoMed project where the target schema is called global schema and it is defined as the integration of the sources. Mappings are called query translation pathways and they are defined between the global schema and each source schema. AutoMed adapts the translation pathways for both the global view and the source evolution. The global schema may be also evolved from the source evolution.



Transformation S1 -> GS
1. addNode <<boss>>{x|<x,T> ∈ <<_,staff,mgrade>>}
2. addCons <<boss>> ⊆ <<staff>>
3. delEdge <<_,staff,mgrade>>
   {x,T| x∈<<boss>>}∪{x,F | x∈<<staff>>-<<boss>>}
4. delCons <<mgrade>> = {T, F}
5. delNode <<mgrade>> {T, F}
6. extendNode <<site>>
7. extendEdge <<located_at, division, site>>

Transformation S2 -> GS
1. renEdge <<within, stuff, dept>> work_in
2. extendNode <<skill>>
3. extendEdge <<_, stuff, skill>>

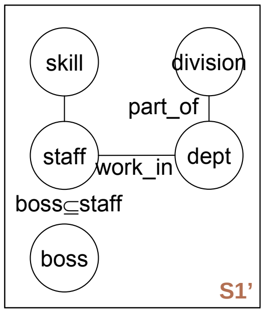
Figure 2-17. Two HDM source schemas and a global schema

In AutoMed, each schema is represented in the hypergraph data model (*HDM*), which is a triple <*Nodes, Edges, Constraints*>. The query translation pathways are expressed over a schema *S* using a set of primitive transformation operators on schemas. Some of them are shown as follows:

– *adding* a node, an edge, or a constraint when this new construct can be derived from the existing constructs of *S*; for example, adding in a schema containing a node "person" a node "professor": "professor" can be derived from "person";
– *extending* a node or an edge when the new construct cannot be derived from *S*;
– *deleting* a node, an edge, or a constraint; In this case, the deleted construct can be reconstructed from the extents of the remaining schema constructs; for example, deleting in a schema containing two nodes "person" and "professor" the node "professor": "professor" can be reconstructed from "person";
– *contracting* a node or an edge when the contracted construct cannot be reconstructed from *S*;
– *renaming* a node or an edge;

Figure 2-17 illustrates three schemas *S1*, *S2,* their integrated global schema *GS* and the transformations from *S1* and *S2* to *GS* respectively. The node *mgrade* in *S1* is constrained to have a two-valued extent {*T, F*} *T* for managers and *F* for other staff.

The approach of mapping adaptation is based on mapping composition. If some source schema S_i evolves, say to S_i*'*, the transformation *t* from S_i to S_i*'* is first specified. It is then combined with the transformation between S_i and *GS* to get the transformation *Ti'* from S_i*'* to *GS*. Suppose that *S1* of Figure 2-17 is changed to *S1'* in Figure 2-18, the transformation from *S1* to *S1'*, denoted by *t*, is shown in Figure 2-18. Given the transformation *T* from *S1* to *GS*, the transformation from *S1'* to *GS* is *T-t* as shown in Figure 2-18.



Transformation S1 -> S1'
1. addNode <<boss>>{x|<x,T> ∈ <<_,staff,mgrade>>}
2. addCons <<boss>> ⊆ <<staff>>
3. delEdge <<_,staff,mgrade>> {x,T| x∈<<boss>>}∪{x,F | x∈<<staff>>-<<boss>>}
4. delCons <<mgrade>> = {T, F}
5. delNode <<mgrade>> {T, F}

Transformation S1' -> GS
1. extendNode <<site>>
2. extendEdge <<located_at, division, site>>

Figure 2-18. Evolution of the source schema S_1 to S_1'

If the new schema does not add components that are not in the global schema or remove components that are not in the other sources, only pathways will be repaired. If added components are not in the global schema yet or deleted components are not in any other source, the global schema may be evolved.

After the operations of contracting source schemas, the global schema *GS* may contain constructs that are no longer supported by any source schema. In this case, the global schema is repaired by contracting these unsupported constructs. Once the global schema is repaired, every related transformation pathway is repaired again by removing every transformation that extends the unsupported constructs.

If a source schema S_i is transformed to S_i*'* by an *extend* transformation and the global schema does not contain this construct yet, the global schema is evolved by adding the new construct. The transformation pathways are evolved afterwards.

AutoMed also considers the changes of the global schema expressed using the same operators. Adapting mappings for target schema changes is similar to the one for source changes by first finding evolution mappings then compose them with the existing one. For the target changes, only transformation pathways will be adapted and no source schema will be changed for the target evolution.

3.3. Discussions

The previous approaches on mapping adaptation assume different contexts and have different objectives. Table 2-2 shows some of their main features such as schema models, mapping characters, whether or not they consider the source and target schema changes, and whether mappings involve joins between different sources. As for mapping generation, mapping adaptation becomes more complicated when mappings express joins between different sources. The volume of the data that the system needs to manage is more important and the system also needs to identify the same objects from different sources. The mappings considered by EVE and Bouzeghoub et al.'03 are of this kind. In AutoMed, ToMAS and MACES, each mapping is defined for one target schema and one source schema. EVE and Bouzeghoub et al.'03 consider only the changes in the source schemas. The other approaches consider changes in both the source or target schema. "Generalizability" shows whether the approaches assume a specific mapping generation process and if their mapping adaptation depends on the way mappings are generated. Bouzeghoub et al.'03 and ToMAS can be considered to be an incremental algorithm of an existing mapping generation approach. MACES and AutoMed both need a mapping generation approach to generate evolution mappings. EVE does not depend on a mapping generation approach.

The remainder of this section will provide some discussions over the changes considered by the approaches, their costs, as well as their context.

Table 2-2. Main features of the different approaches

Approaches	Schema model	Mapping Description Language	Mapping Involving Inter-Source Joins?	Source Evolution?	Target Evolution?	Generalizability
EVE	Relational	SQL	Yes	Yes	No	Generalizable
Bouzeghoub et al.'03	Relational	SQL	Yes	Yes	No	Not generalizable
ToMAS	Nested relational	Specific logical expressions	No	Yes	Yes	Not generalizable
MACES	Nested relational	Specific logical expressions	No	Yes	Yes	Not generalizable
AutoMed	Graph (ER-liked)	Transformation pathways	No	Yes	Yes	Not generalizable

Considered Changes

In this section, we discuss the changes supported in the existing approaches of mapping adaptation. Among the 5 works, there are two of them (EVE and Bouzeghoub et al.'03) that only support source evolution. The others (ToMAS, MACES and AutoMed) support both source evolution and target evolution.

Table 2-3 shows the source schema changes considered by the different systems and the actions to adapt them. The changes comprise adding or removing elements or constraints, schema restructuring and element renaming.

Adding source elements is considered by the approaches EVE, Bouzeghoub et al.'03 and AutoMed. AutoMed adapts the mappings and potentially evolves the target schema. EVE consider the change but since the mapping is not affected the approaches do not propose any adaptation action. Bouzeghoub et al.'03 also considers the change and it adapts the operation graph. ToMAS supports adding source elements but they do not support adding semantic correspondences for the added elements. Since their mappings are based on correspondences, no adaptation can be done. MACES does not support adding source elements. Since all changes are expressed using the evolution mapping between the old schema and the new schema, all components that do not exist in the old schema can not be described in the evolution mapping.

Removing source elements is supported by all these approaches: AutoMed repairs mappings and potentially evolves the target schema; Bouzeghoub et al.'03 and ToMAS track their uses of the removed element in different steps of the mapping generation and repair the corresponding results; EVE and our approach search for substitutions to replace the deleted elements; and MACES adapts the mapping for the changes.

All the approaches, except EVE, consider source constraints changes. These approaches all assume that their mappings should satisfy certain constraints. When constraints are removed or added, mappings need to be adapted. For EVE, constraints are in a source description server and EVE does not support changes inside of the server. All the approaches can directly (ToMAS, MACES, AutoMed and our approach) or indirectly (EVE and Bouzeghoub et al.'03) express source schema restructuring. Renaming is supported by all the approaches.

ToMAS, MACES and AutoMed also support target schema changes which are shown in Table 2-4 For target evolution, the mapping composition approaches (MACES and AutoMed) adapt the mappings in a same way as for source evolution since the adaptation consists in the composition of the original mapping with the evolution mapping. Mappings considered in ToMAS have a symmetric form, so ToMAS adapts mappings for target changes in the same way as it does for source changes.

Table 2-3. Considered source schema evolution of the different approaches

Approaches	Adding elements	Removing elements	Adding/removing constraints	Restructuring subtrees	Renaming elements
EVE	no action	Substitution search	(do not consider)	(do not consider)	Syntactical changes
Bouzeghoub et al. 03	Mapping adaptation	Mapping adaptation	Mapping adaptation	Mapping adaptation	(do not consider)
ToMAS	(do not consider)	Mapping adaptation	Mapping adaptation	Mapping adaptation	Syntactical changes
MACES	(do not consider)	Mapping composition	Mapping composition	Mapping composition	Mapping composition
AutoMed	Mapping composition and target schema evolution	Mapping composition and target schema evolution	Mapping composition	Mapping composition	Mapping composition

Table 2-4. Considered target schema evolution of the different approaches

Approaches	Adding elements	Removing elements	Adding/removing constraints	Restructuring subtrees	Renaming elements
ToMAS	(do not consider)	Mapping adaptation	Mapping adaptation	Mapping adaptation	Syntactical changes
MACES	(do not consider)	Mapping composition	Mapping composition	Mapping composition	Mapping composition
AutoMed	Mapping composition	Mapping composition	Mapping composition	Mapping composition	Mapping composition

Costs

The re-definition of the mapping when the schemas evolve is a time consuming process, especially at runtime. The main reason of adapting the original mappings to the new schemas is to avoid re-generating mappings every time changes occur in the schemas.

Incremental approaches are efficient for the contexts in which schemas change in little steps. Instead of generating the whole mapping, the algorithm only repairs the part of the mapping affected by the change. The time to adapt mappings for one single change is in most cases shorter than from scratch generation. In contexts where changes occur frequently in the schemas, incremental approaches become less efficient. The algorithm has to be executed for every single change. When the sequence of changes is long, the total time for all changes of the sequence may be longer than the time used by a mapping composition approach and the time to directly generate mappings for the new schemas.

Mapping composition approaches first generate evolution mappings then compose them with the original mapping to obtain new mappings. This approach is less efficient than incremental approaches if few changes occur in the system. However, it is more efficient than incremental approaches in an inverse context.

We also compare the performances between the mapping adaptation approaches by mapping compositions with from-scratch mapping generation for new schemas. In a mapping composition approach, if some changes occurred in the system, the evolution mapping is first generated and then composed with the original mapping. Consider that the time for generating the evolution mapping is *Te* and the time for composing mappings is *Tc*, the total time for adapting mapping is *Te + Tc*. Consider the time for generating mappings directly for the new schemas is *Tn*. Using a mapping composition approach is more efficient than from-scratch mapping generation if *Te + Tc > Tn*. Consider that we use the same mapping generation approach (such as Clio'02) to generate both evolution mappings or new mappings. The time for mapping generation often depend on the size (i.e. number of nodes) of the target and source schemas. In the context that the changed schema is more complex than the other one, the time of *Te* can be bigger than *Tn*; then directly generating mappings for the new schemas may be more efficient. Mapping composition approaches become more efficient than from-scratch mapping generation if the size of the changed schema is less than the size of the other schema.

Generalizability

Generalizability is an important feature; more an approach considers the standard languages with minimum assumptions, more it can be used in a large number of existing systems.

As the mapping generation approaches, all the mapping adaptation approaches consider have the target schema and the source schemas. They also assume a set of semantic correspondences are provided. Recall that providing a support to schema matching is a difficult problem and there exist many research works address it [BLN86, DLD04, DR02, Gal06, HC06, RB01 and SE05].

Among the existing approaches on mapping adaptation, some approaches are tightly related to the way their mappings are generated: Bouzeghoub and al.'03 considers mappings generated by Kedad and Bouzeghoub'99; and ToMAS considers mappings generated by Clio'02. These approaches can be only used over their respective mapping generation approach. They consider that intermediate results of the associate mapping generation are provided and permanently maintained; the mapping adaptation approach can be considered as an incremental execution of the mapping generation.

For the mapping composition approaches (MACES and AutoMed), the algorithm of mapping composition does not relate to the way mappings are generated. However, evolution mappings need to be generated. Both of the two approaches consider mappings expressed in a specific language. They therefore need to use a specific mapping generation approach (Clio'02 and AutoMed) to generate evolution mappings.

Unlike all the other approaches, EVE does not rely on a mapping generation approach.

3.4. Limitations of the Existing Approaches

Most of the existing approaches on mapping adaptation are related to a particular mapping generation approach. Among the incremental approaches, both Bouzeghoub et al.'03 and ToMAS are the incremental execution of their corresponding mapping generation approach. The intermediate results of these corresponding mapping generations need to be provided and permanently maintained.

AutoMed and MACES are mapping composition approaches and they are both based on a mapping generation approach to generate evolution mappings. As we discussed before, generating evolution mappings is itself a costly process and it is sometimes even more costly than generating mappings directly between new schemas. Moreover, due to the limitations of the language describing mappings, evolution mappings of MACES cannot express all additional changes such as adding elements or adding referential constraints. Since added objects do not exist in the original schema, they cannot be expressed in the mapping. AutoMed use a series of transformation to express evolution mapping and it can express addition changes. But the schemas of AutoMed are expressed in a highly abstract language and it does not express some information that we usually find in a schema (e.g. attributes and keys, etc.). Mappings of both AutoMed and MACES are expressed in an abstract language and can only be used in a specific system.

The EVE approach does not relate to a mapping generation approach. However, a large amount of metadata is required by this approach such as all the joins that are possible between all the source relations and the relation between instances of different source relations. Providing such metadata is a difficult task; sometimes even unrealistic in some contexts such as distributed systems. The approach does not support changes in the target schema.

3.5. Our Proposal

We propose an incremental approach for mapping adaptation. Given one target schema and several source schemas expressed in XML Schema and a mapping for them expressed through XQuery, it adapts the mapping when the target schema or a source schema evolves. We assume semantic correspondences between the schemas and the result of the adaptation is a set of adapted mappings representing alternative ways to adapt the mapping to the new schemas.

Being an incremental approach, we consider a set of changes that can occur in the target schema and in the sources. In the sources, we consider addition and removal of elements, keys and references and we also consider subtree moving within a schema. In the target schema, we consider addition and removal of elements and keys and we also consider moving of subtrees. For every change, the mapping adaptation consists in three steps:

– the original mapping is first checked to see if it is affected by the change. Not all the changes affect the mapping. For example, a mapping will not be affected by a change occurring in a schema that is not related to the mapping. There are two cases that a mapping can be affected: either the target schema has changed such that the mapping does not satisfy the new needs anymore, or some source resources that are used in the original mapping have been moved or removed.
– the affected mapping is then checked to see if it is adaptable for the new schemas, which means there is at least one solution to adapt it to the new schemas. If there are new needs in the target, the mapping should be extended to support the needs. If some source components used in the mapping have been moved or removed, the substitutions have to be found to replace them.
– if the mapping is adaptable, adapted mappings are generated using the adaptation solutions. Several adapted mappings are generated if there are several solutions.

The approach has a specific adaptation algorithm for every change type. A source element removing affects the mapping using it and finding a solution to repair the mapping consists in finding the substitutions of the removed element. A target element removing affects the mapping populating it and repairing the mapping consists in removing the corresponding assignment from the mapping.

To compare to the existing works of mapping adaptation, we consider the target and the source schemas expressed in XML Schema and we consider mappings defined in XQuery using a specific format. The approach also assumes a set of semantic correspondences to enable mapping adaptation.

The mapping we adapt are mapping that involve joins. As for mapping generation, mapping adaptation becomes more complicated when mappings express joins between different sources. The volume of the data that the system needs to manage is more important and the system also needs to identify the same objects from different sources.

As EVE, our approach does not reply on a mapping generation approach. The mapping can be generated using an automatic approach and it can also be specified manually. Appendix I shows the translations of different XQuery queries into our format.

Unlike EVE, we do not require any extra metadata besides schemas and correspondences, and we support target changes. Differ from MACES that can not express addition changes, we consider both target element addition and source element additions.

4. Conclusions

In this chapter, we presented some existing works on mapping generation and adaptation. Many research approaches have been proposed for automatic mapping generation as well as industrial tools. We showed in this chapter the principle of the research approaches and the functionalities of the industrial tools. Through the analysis of those proposals, we can notice that few of them can generate mappings involving joins between different sources. However such mappings can be necessary in some different contexts such as data integration systems. Only one existing proposal generates mappings involving inter-source joins for XML schemas but it assumes that the target schema and the source schemas are homogeneous. Such assumptions are difficult to realize in today's distributed environment.

We also presented some proposed works for mapping adaptation. Most of those works depend on a specific mapping generation approach. Only one approach does not have such assumption but it requires a large volume of source descriptions and some of them are difficult to be obtained.

As shown, we address the problem of automatic mapping generation and adaptation for XML schemas. We focus on generating mappings from multiple source schemas and mapping can express inter-source joins. We do not assume any homogeneity between the target schema and the source schemas and we generate mappings in an abstract language and can be later on translated in another more declarative language. We adapt mappings expressed in XQuery when schemas evolve. The mappings can be generated by a tool or manually specified by a designer and the mapping adaptation does not depend on a mapping generation process. The approach does not require any extra metadata besides schemas and correspondences.

Chapter 3. Automatic Mapping Generation for XML Data Sources

1. Introduction

Information exchange between different applications and the integration of distributed heterogeneous data sources are becoming increasing needs in today's information systems. In this context, every data provider exports a schema, called ***source schema***, to describe the data it is willing to share with external applications. Every application that asks for information provides a schema, called ***target schema***, to express its information needs. The way to transform the data from the source representations to the target representation defined by the application is expressed using a ***mapping***.

A mapping is an expression defining how instances of a given target schema can be obtained for a set of sources. Mappings can be used in many different contexts such as data translation systems, wrappers, mediation systems, or data warehouses. In data translation systems and wrappers, mappings usually relate one target schema to only one source schema. They express the transformation of the data from the source representation to the target representation. In the data integration systems like mediation systems and data warehouses, there are often several sources and the application requires the integration of the sources data. In this case, mappings may be defined for the target schema from several source schemas. In a mediation system where the target schema is called mediation schema and mappings are called mediation queries, mappings are used to rewrite user queries expressed over the target schema in terms of the source schemas.

Mapping generation is usually considered to be a manual and extremely difficult process. The designer of the system must have a thorough understanding of not only the target schema and all the data sources, but also the semantic links between the sources and the target schema. Especially when the number of data sources is important, the amount of metadata to manage makes a manual mapping generation a tedious and time consuming process. Furthermore, the evolutions occurring in the system may require the redefinition of the mappings. If a source changes its schema, the mappings may become obsolete and need to be redefined.

The problem of automatic or semi-automatic mapping generation has recently become an active research area and many works have been proposed in this subject. In Chapter 2, we present a survey of the approaches that generate mappings automatically or semi-automatically. [CLL03, FS03, FW06, KB99, MHH00, PVM02, SKR01, and Zam04]. Among these works, some generate mappings between relational schemas [FW06, KB99 and MHH00]. Following the proposal of XML (eXtensible Markup Language [xml]) that is particularly well-suited common standard for data exchange between applications, other works [CLL03, FS03, PVM02, SKR01, and Zam04] and some industrial tools [AIS, AMF and Sty] have been proposed to generate mappings between XML schemas. Most of the proposed works generate mappings between one target schema and one source schema. Two approaches [KB99 and FS03] generate mappings for one target schema from several source schema and their result mappings can express joins between different schemas. Some approaches [CLL03, FS03 and SKR01] generate mappings between source and target schemas that have a homogeneous structure, while the others do not make any assumption about the homogeneity of these structures.

There are still some open problems in mapping generation; only two of the existing approaches [KB99 and FS03] generate mappings involving joins between different sources. [KB99] address the problem of mapping generation between relational schemas and [FS03] generates mappings involving inter-source joins between schemas expressed using XML schema. However, it assumes that the target schema and the source schemas have a homogeneous structure, and the mappings are generated in a specific language

Our goal is to propose an approach to support the process of mapping generation. Compared to the existing works, our approach generates mappings for one XML target schema from several XML data sources and the generated mappings may involve inter-source operations. The result of the process is a set of mappings representing alternative semantics; the result mappings are specified in an abstract format and can be translated into a given language such as XQuery. If the target schema can not be satisfied by the sources, the generation process proposes to relax some constraints in the target schema such that a mapping can be found.

The basic idea our automatic mapping generation approach is to (i) first decompose the target schema into subtrees, called ***target subtrees***; (ii) then define mappings, called ***partial mappings***, for these subtrees independently from the others; (iii) finally combine partial mappings of different subtrees to generate the mappings for the whole target schema. At the end of the process, if no mapping has been generated, this means that the information required by the target schema cannot be provided by the sources. We propose in this case a procedure for relaxing either some cardinality constraints or some structural constraints defined in the target schema in such a way that a mapping can be generated for the new target schema. This process requires user interaction to either validate a new target schema after constraint relaxation or choose between different options of relaxation.

The remainder of the chapter is organized as follows. In Section 2, we present some basic assumptions and preliminary definitions that are used in our approach. Section 3 presents an overview of automatic mapping generation process. Sections 4, 5 and 6 describe respectively the main three steps of the mappings generation: the target schema decomposition, the partial mapping definition, and the target mapping generation. The algorithms used in the generation are also presented. Section 7 presents our adaptation of the mapping generation approach to handle the cases where the target schema cannot be satisfied by the sources by relaxing some constraints in the target schema. Section 8 concludes the chapter.

2. Basic Assumptions

To carry out the process of mapping generation, our approach uses some metadata describing the source and the target representations as well as the semantic links between them. We suppose that the target schema and the source schemas are available and we suppose that a set of semantic correspondences between the schemas is provided.

We consider that both the target schema and the source schemas are expressed using XML Schema [xsd]. There are two languages to define XML documents: DTD (Document Type Declaration [dtd]) and XML Schema [xsd]. Both of the two languages allow defining structure, content and constraints (key and references) for XML sources. XML Schema includes the full capabilities of DTDs and it can also express more information such as relative keys (keys that are valid only in a portion of the documents) that cannot be expressed in DTD. Consequently, we made the choice of XML Schemas as the representation of the schemas in our mapping generation approach.

We also assume that a set of semantic correspondences is provided between the target schema and the source schemas. They will be used in our approach to generate mappings. Producing such semantic correspondences is an old problem known as schema integration or schema matching

[BLN86] and it is still an active research area [DLD04, DR02, Gal06, HC06, RB01 and SE05]. We suppose that the semantic correspondences used in our approach are given; and generating such semantic correspondences is out of the scope of this thesis. The quality of the result mappings depends on the quality of the input metadata. Consider the semantic correspondences that are used to generate mappings. If the set of correspondences used by the system is incomplete, then some possible solutions may be missing in the result set of mappings.

In the remainder of the section, we first present in Section 2.1 the representation of the different schemas and then in Section 0 the semantic correspondences considered in our approach.

2.1. Schemas

We consider that the source schemas and the target schema are expressed using XML Schema [xsd]. This language allows defining structure, content and constraints (key and references) for XML sources. Figure 3-1 depicts two source schemas and a target schema representing information about books in a library. The representations of these three schemas in XML Schema [xsd] are described in Appendix I. To avoid confusions, each element will be suffixed by the name of its schema: *authorid*$_{s1}$ will refer to the element author-id in S1, while *isbn*$_{s2}$ will refer to the element isbn in S2. Every element in the tree may be either a text element (e.g. *authorid*$_{s1}$), that is, an element containing only text, or an internal element (e.g. *chapter*$_{s1}$). The leaves of the tree are always text elements.

The cardinality of every element is characterized by the attributes `minOccur` and `maxOccur`, representing respectively the minimum and maximum number of instances for this element in the tree with respect to its parent. Each element is monovalued (`maxOccurs` = 1) or multivalued (`maxOccurs` > 1); it is also optional (`minOccurs` = 0) or mandatory (`minOccurs` > 0). In Figure 3-1, the symbol '+' represents a multivalued and mandatory element (e.g. *book*$_{s2}$); the symbol '*' represents a multivalued and optional element (e.g. *book*$_{ts}$); and the symbol '?' represents a monovalued and optional element (e.g. *abstract*$_{ts}$). An element without any symbol behind is monovalued and mandatory (e.g. *id*$_{s1}$).

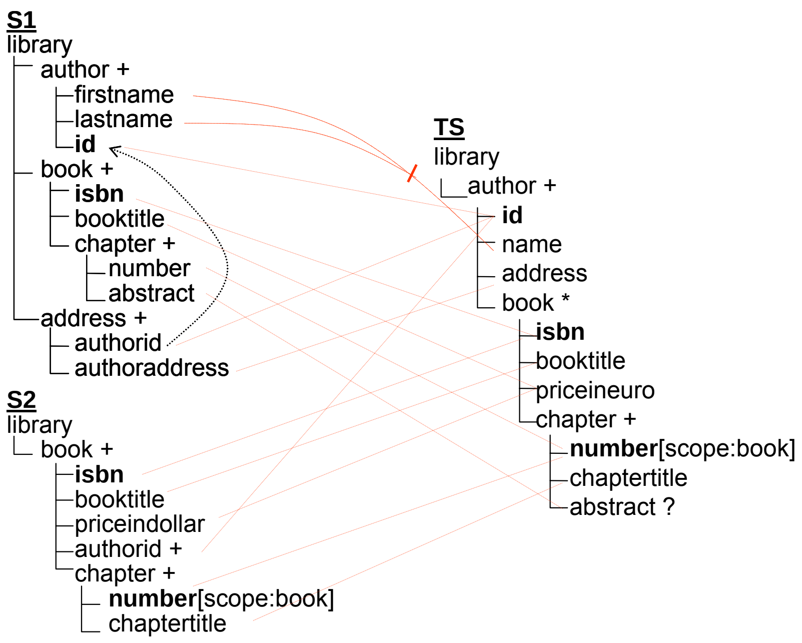


Figure 3-1. Schemas and correspondences

Keys are defined either in the whole schema or only in a subtree of the schema. In the former case, the key is absolute. In the latter case, the key is relative and its scope is an antecessor of the identified element, except the root. In Figure 3-1, the elements written in bold represent keys. If the name of a key element is followed by a bracket, then the key is a relative key and its scope is the element between brackets (e.g. $number_{s2}$ is a relative key and its scope is $book_{s2}$), otherwise it is an absolute key (e.g. $isbn_{s1}$). A schema may also contain references; each one is a set of text elements referencing another set of text elements defined as a key. In our example, $authorid_{s1}$ references id_{s1}, and this is represented by an arrow in Figure 3-1.

2.2. Semantic Correspondences

Semantic correspondences between elements of different schemas state that these elements have the same meaning. We assume that a set of semantic correspondences is provided between the target schema and the source schemas and our goal is to provide support for generating a mapping definition using these semantic correspondences. We distinguish between three kinds of correspondences: simple 1-1 correspondences, 1-1 correspondences involving a transformation function and 1-n correspondences.

- ***a 1-1 correspondence*** relates one element *n* in the target schema to one element *n'* in a source schema and states that the two elements represent the same concept. We denote this correspondence $n \cong n'$. (e.g. $id_{s1} \cong id_{ts}$ and $number_{s2} \cong number_{ts}$);
- ***a 1-1 correspondence involving a transformation function*** relates one element *n* in the target schema to one element *n'* in a source schema and a transformation function *f* is applied to *n'*. It states that *n* represents the same concept as *n'* after applying the transformation function and it is denoted $n \cong f(n')$. For example, a correspondence can relate the target element $priceineuro_{ts}$ to the source element $priceindollar_{s2}$: $priceineuro_{ts} \cong con(priceindollar_{s2})$ where the function *con* is the conversion from dollar to euro following a given exchange rate.
- ***a 1-n correspondence*** relates one target element *n* to a set of source elements $n_1, .., n_k$ combined using a transformation function $f(n_1, .., n_k)$. It states the equivalence between *n* and $f(n_1, .., n_k)$ and it is denoted $n \cong f(n_1, .., n_k)$. For example, we can have a 1-n correspondence to relate the target element $name_{ts}$ with the concatenation of the source elements $firstname_{s1}$ and $lastname_{s1}$: $name_{ts} \cong concat(firstname_{s1}, lastname_{s1})$. The 1-n correspondences can be seen as the general form of all the previous correspondences.

Previous work has shown how semantic correspondences can be defined automatically. [DLD04, DR02, Gal06, HC06, RB01 and SE05]; we suppose that these correspondences are provided and we do not address the problem of generating them. In the example of Figure 3-1, dotted lines between the target schema and the source schemas represent correspondences.

We use the same notation to represent correspondences between sets of elements. Given a set a set of target elements *S* and a set of source elements *S'*, there is a correspondence between *S* and *S'* if the follows conditions are satisfied:

- for each element *n* in *S* there is exactly one subset *N'* of *S'* such that $n \cong f(N')$;
- every element in *S'* is involved in exactly one correspondence with an element of *S*.

In the example of Figure 3-1, there are three semantic correspondences: $id_{ts}, \cong id_{s1}$, $name_{ts} \cong concat(firstname_{s1}, lastname_{s1})$, and $booktitle_{ts} \cong booktitle_{s1}$. We can therefore consider that there is a semantic correspondence $\{id_{ts}, name_{ts}, booktitle_{ts}\} \cong \{id_{s1}, firstname_{s1}, lastname_{s1}, booktitle_{s1}\}$.

Correspondences between two source schemas are derived through their correspondences with the target schema. Given two source elements *n* and *n'* in two different source schemas, the correspondence $n \cong n'$ holds if there is an element n'' in the target schema such that $n'' \cong n$ and $n'' \cong n'$. The same holds for two sets of source elements N and N' in two different source schemas. There is a correspondence $N \cong N'$ if there is a set of target elements N'' such that $N'' \cong N$ and $N'' \cong N'$. Some correspondences may also be provided between the source schemas. In this case, they will be used in our approach for mapping generation. For example, in the scenario considered in the MediaGRID project[1], there are three biological data sources *GOLD*, *SMD* and *SGD*. The domain experts are able to state that the element *gname* from *SMD* represents the same concept as the element *gene* in the source *SGD* (i.e. *gname* $\cong$ *gene*) even if such elements have no equivalent in the target schema.

3. An Overview of our Automatic Mapping Generation Approach

This chapter gives an overview of our proposal on automatic mapping generation. Given one target schema and a set of source schemas in XML Schema, our approach generates a set of mappings to derive instances of the target schema from the instances of the sources. Similarly to all the existing approaches, it considers a set of semantic correspondences between the schemas. Applying the generation process creates a set of mappings, each of them representing a different way to define the target from the sources.

To handle the complexity of mapping generation, our approach comprises three steps:

- it first decomposes the target schema into a set of subtrees, called **target subtrees**; the decomposition is done in such a way that all the elements in each target subtree, except the root, are monovalued.
- then it defines mappings for each target subtree, called **partial mappings**; each partial mapping defines a way to derive instances of a target subtree from the instances of the source schemas; there may be several partial mappings for a target subtree to describe different ways to derive its instances. Since a target subtree contains only monovalued elements except the root, finding a mapping for this subtree consists in finding some equivalent elements in the sources regardless their hierarchical organization. The process of defining partial mappings is inspired by Kedad and Bouzeghoub'99 [KB'99, BKA'04] and it uses the semantic correspondences and the key and the referential constraints defined in the sources.
- Finally, partial mappings for different target subtrees are combined to generate mappings for the whole schema, called **target mappings**. Every combination needs to satisfy the parent-child relationships existing between the target subtrees to lead to a target mapping.

The generated mappings are expressed in an abstract language and they can be translated into a given query languages. We propose a translation algorithm to translate the abstract queries into XQuery [xquery]. Figure 3-2 shows the general framework of our mapping generation approach.

We will illustrate the principle of our mapping generation through the example of Figure 3-3. There is one target schema, two source schemas and some semantic correspondences that relate the elements of the target schema to elements of the source schemas.

The target schema is first decomposed into two target subtrees *t1* and *t2*. The two target subtrees are shown in Figure 3-3. In *t1*, except the root $prof_{ts}$, all the other the elements are monovalued.

[1] The MediaGRID Project, a mediation framework for a transparent access to data sources: http://www-lsr.imag.fr/mediagrid/

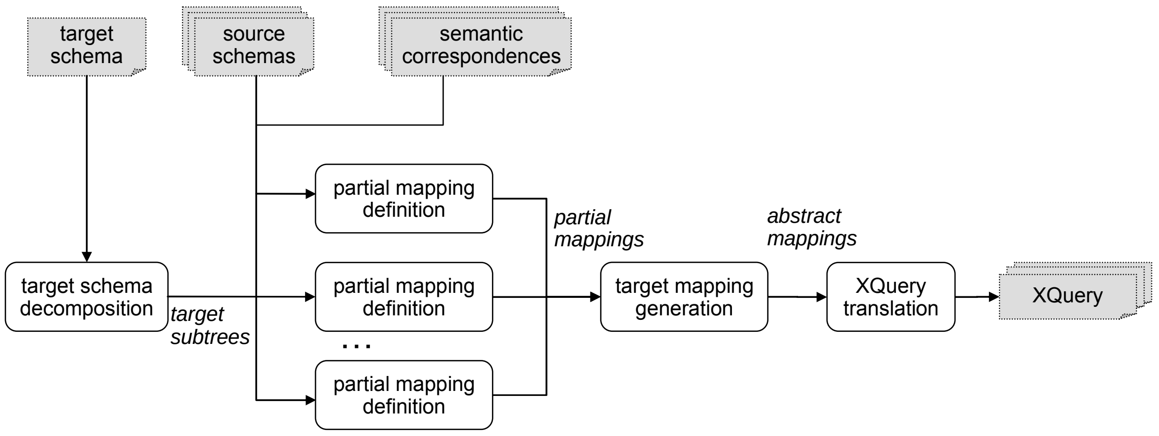
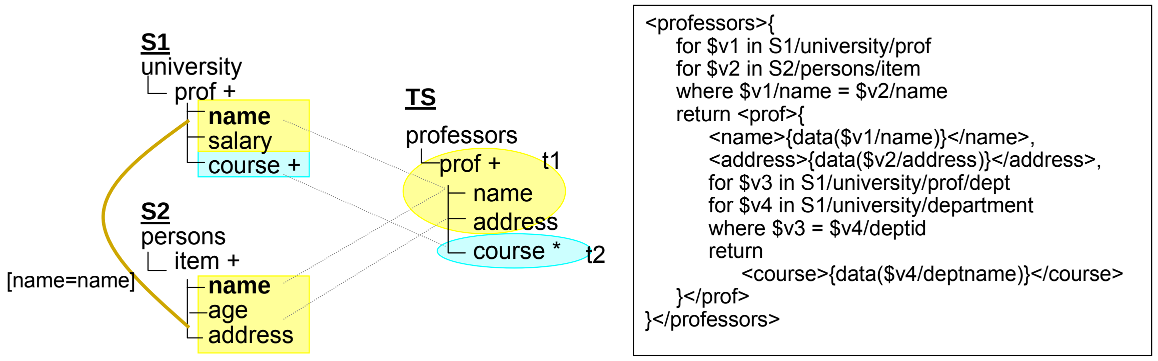

target schema
source schemas
semantic correspondences
partial mapping definition
partial mapping definition
partial mapping definition
target schema decomposition
target subtrees
partial mappings
abstract mappings
target mapping generation
XQuery translation
XQuery

Figure 3-2. The general framework of our mapping generation approach

For each target subtree, partial mappings are defined. For the example in Figure 3-3, there are two partial mappings *pm1* and *pm2* for *t1*. The partial mapping *pm1* defines *t1* from the elements $name_{s2}$ and $address_{s2}$ of *S2*. The partial mapping *pm2* joins *S1* and *S2* with the join predicate [$name_{s1} = name_{s2}$] and defines *t1* using $name_{s2}$ and $address_{s2}$ from the result of the join. These partial mappings represent two alternative ways to derive instances of *t1*: compared to *pm1*, *pm2* derives fewer instances because of the join but it assures the derived instances occurred in both *S1* and *S2*. There is only one partial mapping *pm3* for the target subtree *t2*: it defines *t2* from $course_{s1}$ in *S1*.

Partial mappings of the two target subtrees are used to generate target mappings. There are two possible combinations: the first one combines *pm1* for *t1* with *pm3* for *t2* and the second one combines *pm2* for *t1* with *pm3* for *t2*. There is one parent-child relationship between *t1* and *t2* to express the link between each professor and the courses he teaches. The two combinations therefore need to satisfy this relationship to be a valid target mapping. This relationship is found in the source S1 in which the courses are organized by their professors. Consequently, only the second combination leads to a target mapping because it defines *t1* from the combination of *S1* and *S2* and it can therefore identify for every returned instance of *t1*, the corresponding instances of *t2*. It is translated into XQuery which gives the result mapping at the right side of Figure 3-3.



```
<professors>{
    for $v1 in S1/university/prof
    for $v2 in S2/persons/item
    where $v1/name = $v2/name
    return <prof>{
        <name>{data($v1/name)}</name>,
        <address>{data($v2/address)}</address>,
        for $v3 in S1/university/prof/dept
        for $v4 in S1/university/department
        where $v3 = $v4/deptid
        return
            <course>{data($v4/deptname)}</course>
    }</prof>
}</professors>
```

Figure 3-3. An example of our mapping generation approach

If the data sources do not allow generating a mapping that satisfies users' needs as represented by the target schema, our approach proposes to relax some cardinality constraints or structural constraints in the target schema such that a mapping can be found for the new schema. For example, a target schema may require information about authors such as names and ages while no source of the environment provides this information and as a consequence, no mapping can be produced to satisfy the target schema. In this case, a solution would be to relax the cardinality constraint of the element defining author ages to optional; a mapping can therefore be produced to derive instances only for author names. This process is interactive and users will review the proposed schemas and choose between alternative solutions.

The general framework of our mapping generation approach has been presented in [KX05b]. In [KX05c and KX05d], we have presented our generation approach using 1-1 correspondences and some algorithms used in the approach. [KX05a] extends the approach by considering 1-n correspondences.

4. Decomposing Target Schemas

To handle the complexity of mapping generation, we decompose the target schema into a set of subtrees, called target subtrees; we first find mappings for each target subtree then combine these mappings to generate the mappings for the whole schema. Given a target schema *T*, we define the target subtrees in *T* as follows:

Definition 3-1: Target Subtrees. Each target subtree t of a target schema *T* is a subtree of *T* satisfying the following conditions:

- the root *r* of the subtree is either a multivalued element or the root of *T*;
- all the other elements in t are descendents of *r* and are monovalued;
- there is at least one text element in *t* (*t* may contain a single element).

This decomposition of the target schema gives several subtrees in which every element is monovalued except the root. The mapping generation problem for the target schema is decomposed into two steps: first finding mappings for every target subtree then combining these mappings. Since a target subtree contains only monovalued elements except the root, finding a mapping for this subtree consists in finding some equivalent elements in the sources regardless their hierarchical organization. The hierarchical structure of the different target subtrees is checked during the second step.

The target schema given in Figure 3-1 has three target subtrees shown on the right side of Figure 3-4: t1 is composed of the multivalued element $author_{ts}$ and its three monovalued children id_{ts}, $name_{ts}$ and $address_{ts}$; t2 is composed of $book_{ts}$ and its three monovalued children $isbn_{ts}$, $booktitle_{ts}$ and $priceineuro_{ts}$; and t3 is composed of $chapter_{ts}$, $number_{ts}$, $chaptertitle_{ts}$ and $abstract_{ts}$. The root $library_{ts}$ does not belong to any target subtree.

Given two target subtrees *t* and *t'* such that the root of *t'* is a child of an element in *t*, we say that *t* is the parent of *t'* and *t'* is a child of *t* (e.g. in Figure 3-4, *t2* is the child of *t1* and the parent of *t3*).

A target subtree can be either mandatory or optional in a target schema. Consider a target schema with the root R and a subtree t of this schema with the root *r*. If *t* has a parent subtree *t'* with the root element *r'*, we say that *t* is mandatory if all the elements on the path from *r* to *r'* (except *r'*) are mandatory. If *t* has no parent subtree, it is mandatory if all the elements on the path from *r* to *R* are mandatory. In all the other cases, *t* is optional. In our example, *t1* and *t3* are mandatory and *t2* is optional.

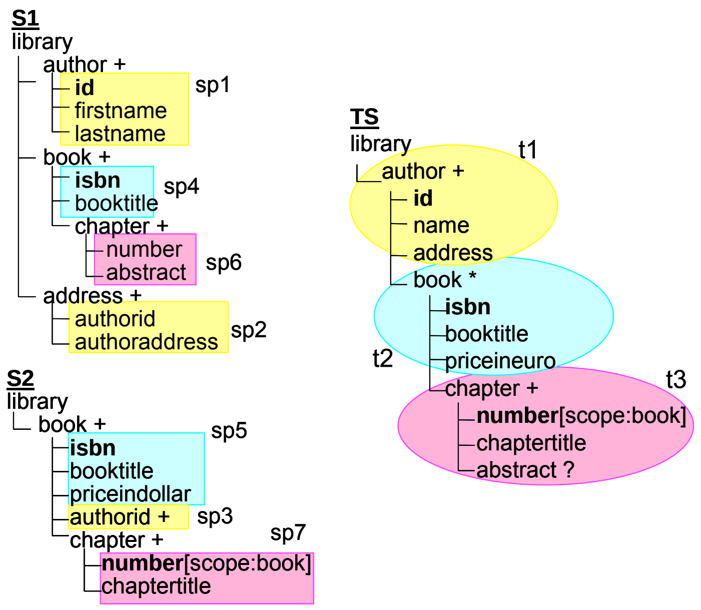


Figure 3-4. Target subtrees and source parts

Algorithm 3-1 describes the target schema decomposition. It takes the target schema (*TS*) as input and produces a set of target subtrees (*T*) in *TS*. The algorithm performs a top-down browsing of the target schema to identify all the subtrees satisfying the conditions.

Algorithm 3-1. Target schema decomposition

```
Target_Schema_Decomposition(TS, T)
Begin
    T := ∅;
    for each element n in TS from top to down:
        if multivalued_or_root(n) //returns true if n is either multivalued or the root of TS
        then
            D := get_monovalued_descendent_text(n);
                //returns the descendents of n that are text elements
            if D ≠ ∅
            then
                t := create_Subtree({n} ∪ D);
                    //create a target subtree with the root n and the elements in D
                T := T ∪ {t};
End
```

5. Defining Partial Mappings

Each partial mapping represents a way to derive instances of a target subtree from the instances of the source schemas. The partial mappings of a given target subtree are defined independently from the other subtrees in three steps represented in Figure 3-5:

- identifying the parts of the sources (called **source parts**) that are relevant for the considered target subtree;
- searching the joins to combine these source parts;
- and defining the partial mappings from the source parts and the joins between them; each target subtree may have several partial mappings.

These three steps are described in the remaining of the section.

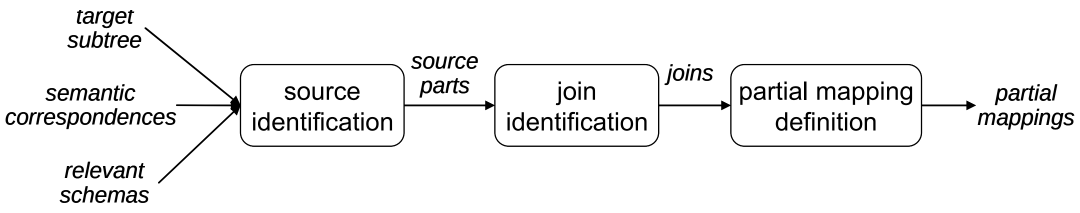


Figure 3-5. The process of partial mapping definition

5.1. Identifying Source Parts

A source part of a given target subtree is a set of text elements in the source schemas that can contribute to derive instances for this target subtree.

Before defining source parts, we present an extended definition of element cardinality. In XML Schema, the cardinality of an element is given with respect to the parent element: an element is multivalued or monovalued with respect to its parent. We generalize this definition to any pair of elements.

Definition 3-2: Extended definition of Cardinality. Given two elements *n* and *n'* in a schema and their first common antecessor *m*, *n* is monovalued with respect to *n'* if every element on the path from *m* to *n* (except *m*) is monovalued. Otherwise, *n* is multivalued with respect to *n'*.

According to this definition, $isbn_{s1}$ is monovalued with respect to $booktitle_{s1}$: their common antecessor is $book_{s1}$ and the only element on the path from $book_{s1}$ to $isbn_{s1}$ (except $book_{s1}$) is $isbn_{s1}$, which is monovalued. Similarly, $booktitle_{s1}$ is monovalued with respect to $isbn_{s1}$. The element $number_{s1}$ is multivalued with respect to $isbn_{s1}$ because their common antecessor is $book_{s1}$ and the path from $book_{s1}$ to $number_{s1}$ contains $chapter_{s1}$ that is multi-valued. On the contrary, $isbn_{s1}$ is monovalued with respect to $number_{s1}$.

Note that this extended definition of cardinality is different from the definition of functional dependency. Consider the elements $chaptertitle_{s2}$ and $number_{s2}$ in *S2*. The element $chaptertitle_{s2}$ is monovalued with respect to $number_{s2}$. However, the functional dependency $number_{s2} \rightarrow chaptertitle_{s2}$ does not hold as we can see in Figure 3-6: two different instances of chapter number may have the same value, but associated with different titles. In fact, there are several titles for a given chapter number, one for each book.

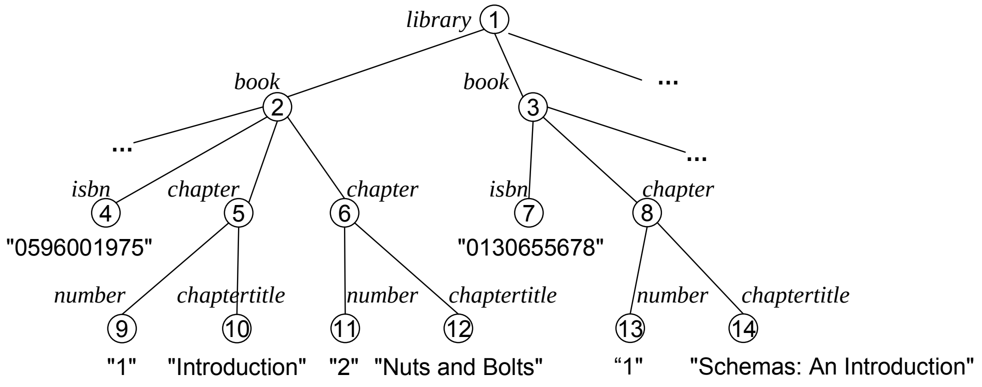


Figure 3-6. An example of instances for the source S2

The source parts are defined using this extended definition of cardinality as follows.

Definition 3-3: Source Parts. Given a target subtree *t*, a source part *sp* for *t* in the source schema *S* is a set of text elements that satisfies the following conditions:

- there is a set of text elements *c* in *t* such that $c \cong sp$;
- there is one element *n* in *sp* such that the other elements in *sp* are monovalued with respect to *n*;
- there is no other set of text elements *c'* in *S* such that $sp \subseteq c'$ and *c'* satisfies the two above conditions.

Given a target subtree *t*, every source element involved in a correspondence with the elements of *t* is found in at least one source part for *t*. If there is no source part, there is no relevant element for *t* in the sources.

Consider the target subtree *t1* of Figure 3-4 having the text elements id_{ts}, $name_{ts}$ and $address_{ts}$. In the set $\{id_{s1}, firstname_{s1}, lastname_{s1}\}$, all the elements are monovalued with respect to the others; this set is therefore a source part for *t1*. In $\{authorid_{s1}, authoraddress_{s1}\}$ both $authorid_{s1}$ and $authoraddress_{s1}$ are monovalued with respect to the others; this set is also a source part for *t1*. $\{id_{s1}\}$ is not a source part because it is a subset of $\{id_{s1}, firstname_{s1}, lastname_{s1}\}$. $\{authorid_{s1}, authoraddress_{s1}, id_{s1}\}$ is not a source part also because id_{s1} is multivalued with respect to both $authorid_{s1}$ and $authoraddress_{s1}$ and both $authorid_{s1}$ and $authoraddress_{s1}$ are multivalued with respect to id_{s1}.

The source parts for the target subtrees of our example are shown on the left side of Figure 3-4. The subtree *t1* has two source parts *sp1* and *sp2* in *S1* and one source part *sp3* in *S2*; *t2* has two source parts *sp4* and *sp5* in *S1* and *S2* respectively; and *t3* has two source parts *sp6* and *sp7*.

Algorithm 3-2 describes the identification of source parts. It takes as input a target subtree *t*, a source schema *S* and it produces the set of all the source parts (*P*) for *t* in *S*. The algorithm first identifies sets of source elements such that each set contains at most one source element involved in a correspondence with a target element in *t*. It then checks the cardinalities between the source elements of each set to identify the source parts for *t*.

Algorithm 3-2. Source part identification

```
Source-part-Identification(t, S, P)
Begin
    P := ∅;
    Q := ∅; //a collection of sets of elements that are candidates to be source parts
    for each element n in t:
        if text_element(n) //returns true if n is a text element
        then
            if Q == ∅
                for each correspondence n ≅ f(N) in get_correspondences(n, S)
                            //return correspondences relating n with source elements in S
                    Q := Q ∪ {N};
            else
                for each set q in Q
                    for each correspondence n ≅ f(N) in get_correspondences(n, S)
                        q’ := q ∪ N;
                        Q := Q ∪ {q’ };
                    Q := Q - {q};
    while   there is a set q in Q and two elements n and n’ in q such that
            self-multivalued(n, n’)
            //returns true if both n and n’ are multivalued with respect to the other
        q1 := q – {n};
        q2 := q – {n’};
        Q := Q – {q};
        Q := Q ∪ {q1} ∪ {q2};
    for each set q in Q
        if ∃q’ ∈ Q such that q ⊆ q’
        then Q := Q – {q};
    P := Q;
End
```

5.2. Identifying Join Operations

Once the source parts are defined for a given target subtree, the candidate joins between the source parts need to be identified. Such identification uses the semantic correspondences existing between the source elements, the key definitions and the referential constraints of the data sources. We distinguish between two distinct cases: either the two source parts belong to the same source schema or to different ones.

Identifying joins between source parts of the same schema. The identification of joins between source parts of the same schema is based on the key and the key references defined in the schema. Given two source parts *sp* and *sp’* in the same source schema, a join with the predicate *c = c’* is a candidate join if there are two sets of text elements *c* and *c’* in the schema such that:

- *c* is a key and *c’* references *c*;
- there is an element *n* in *c* such that every element in *sp* is monovalued with respect to *n*;
- there is an element *n’* in *c’* such that every element in *sp’* is monovalued with respect to *n’*.

The join is denoted as *j[c = c’](sp, sp’)*. Given the source parts of our example (Figure 3-4), $j[id_{s1} = authorid_{s1}](sp1, sp2)$ is a candidate join between *sp1* and *sp2* since $authorid_{s1}$ is a reference to id_{s1}.

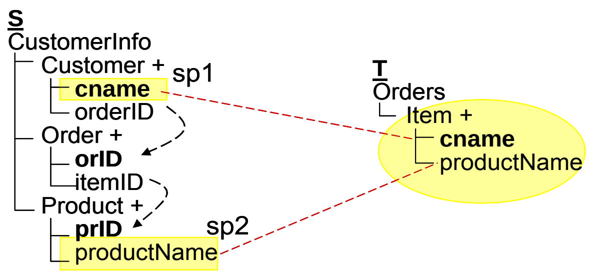


Figure 3-7. Relating two source parts through several references

This definition can be generalized to incorporate a sequence of references from *c'* to *c*. Consider the example shown in Figure 3-7. In the source S, the two source parts *sp1* and *sp2* correspond to the single subtree of the target schema and there is no candidate join between them using the previous rule because no reference relates them directly. However, they are related through the two references: $orderID_s$ referencing $orID_s$ and $itemID_s$ referencing $prID_s$. We generalize the previous definition to consider this case and there is therefore a candidate join between *sp1* and *sp2* denoted by *j[orderIDs1 = orIDs1, itemIDs1 = prIDs1](sp1, sp2).*

Algorithm 3-3 describes join identification between source parts of the same source. It takes a target subtree *t* and a source schema *S* as input and produces a set of candidate joins *J* between the source parts for *t* in *S*. The basic idea of the algorithm is first to browse the references and the referenced keys in *S* to identify the candidate join predicates, then to browse the source parts in *S* to identify the ones that can be related through a join. The way the algorithm identifies joins between source parts of the same schema consists in exploring the referential constraints defined in the schema in the same way as the chase algorithm proposed in [MMS79 and PT99].

Algorithm 3-3. Join identification between source parts of the same source

```
Single_Source_Join_Identification(t, S, J)
Begin
    J := ∅;
    for each r in get_reference_elements(S)
        //returns all the sets of elements defined as references in S
        k := get_refed_key(r); //get the set of elements defined as the key referenced by r
        for each source part p in get_src_parts(t, S) //return the source parts for t in S
            if monovalued(r, p)
                //true if there is an element in r with respect to witch all elements in p are monovalued
            then
                for each source part p' in get_src_parts(t, S)
                    if monovalued(k, p')
                    then
                        j := create_join(p, p', r, k);
                                //creates a join between p and p' with the predicate r = k
                        J := J ∪ {j};
End
```

Identifying joins between source parts of different schemas. If two source parts belong to different schemas, it is not possible to identify candidate joins using some referential constraints since such constraints are defined within a single schema. Consider the book chapters in our example: *S1* contains the number and an abstract for every chapter and *S2* contains the number and the title of every chapter. The only way to obtain the abstract and the title for every chapter is therefore to find a join involving the two sources *S1* and *S2*.

We infer joins between source parts of different source schemas based on the definition of keys and correspondences between elements. Consider two source parts *sp* and *sp'* in the source schemas *S* and S' respectively. Given a set of text elements *c*={*n1, .., nk*} in *S* and a set of text elements *c'*={*n1', .., nk'*} in *S'*, there is a candidate join between *sp* and *sp'* if the following conditions hold:

- $c \cong c'$;
- either *c* or *c'* is an absolute key in its schema;
- there is an element *n* in *c* with respect to which every element in *sp* is monovalued;
- there is an element *n'* in *c'* with respect to which every element in *sp'* is monovalued.

If we only consider 1-1 correspondences, the join predicate is *n1=n1', .., nk=nk'*, and the join is denoted by *j[n1=n1', .., nk = nk'](sp, sp')*. If we consider 1-n correspondences, the join predicate will combine the transformation functions involved in the correspondence $c \cong c'$.

Figure 3-8 shows all the candidate joins in our example; $j[id_{s1} = authorid_{s2}](sp1, sp3)$ is a candidate join between *sp1* and *sp3* because id_{s1} is defined as absolute key. There is no candidate join between *sp6* and *sp7* with the predicate $number_{s1} = number_{s2}$ because neither $number_{s1}$ nor $number_{s2}$ are defined as absolute keys. However, we know that the combination $\{number_{s2}, isbn_{s2}\}$ is unique in the whole schema because the scope of $number_{s2}$ is $book_{s2}$ that has the absolute key $isbn_{s2}$. We have therefore to consider the combination $\{number_{s2}, isbn_{s2}\}$ as an absolute key and use it instead of $number_{s2}$. In fact, each time a relative key is found, it is combined with other key elements to get an absolute key if possible.

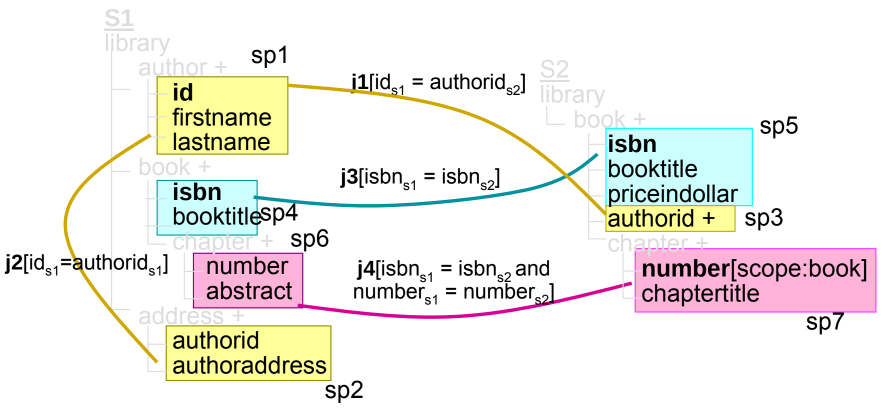


Figure 3-8. Join operations

Algorithm 3-4 describes join identification between source parts of different sources. It takes as input a target subtree *t*, and two source schemas *S* and *S'* and it produces a set of joins between the source parts for *t* in *S* and the source parts for *t* in *S'*. The algorithm first browses the keys defined in *S* and *S'* to identify all the candidate join predicates. Then it browses the source parts in the two schemas to identify the ones that can be related through a join.

Algorithm 3-4. Join identification between source parts of different sources

```
Multi_Source_Join_Identification(t, S, S', J)
Begin
    P := ∅; //the sets of candidate join predicate between source parts in S and S'
    for each k in get_absolute_keys(S)
        //returns all the sets of elements defined as absolute keys in S
        for each k≅n in get_correspondences(k, S')
            //returns the correspondences involving k with elements in S'
            P := P + {(k, n)};
    for each k' in get_absolute_keys(S')
        for each n≅k' in get_correspondences(k', S')
            if (n, k') ∉ P
            then P := P + {(n, k')};

    J := ∅;
    for each pair (C, C') in P
        for each source part p in get_src_parts(t, S) //return the source parts for t in S
            if monovalued(C')
                //true if there is an element in C' with respect to which the others are monovalued
              and monovalued(C, p)
                //true if there is a source element in C with respect to which
                //all the elements in p are monovalued
            then
                for each source part p' in get_src_parts(t, S')
                    if monovalued(C') and monovalued(C', p')
                    then
                        j := create_join(p, p', C, C');
                        //create a join between p and p' with the equal predicate between C and C'
                        J := J ∪ {j};
End
```

In our approach, we consider candidate joins in a limited number of cases; therefore we do not generate all the joins but only a subset of them. For example, a join involving two different sources is considered as a candidate only if the join predicate involves an absolute key. We could also have considered that there is a candidate join every time a correspondence is found between two sets of elements, regardless the key definitions. But in our opinion, the semantics of such operations is not clear and we therefore do not consider them. Consequently, only a subset of all the possible target mappings is generated in our approach.

5.3. Defining Partial Mappings from the Source Parts and the Joins

The partial mappings of a target subtree are defined using the corresponding source parts and the joins between them.

The source parts and the joins corresponding to a given target subtree can be represented by a graph called a join graph where every element is a source part and every edge between two source parts is a join between them. The edges are numbered and labeled with the join predicate. The partial mappings are defined using join graphs as follows:

Definition 3-4: Partial Mappings. Given the join graph of the target subtree *t*, each partial mapping for *t*, denoted *pm*, is a connected acyclic sub-graph *G* of the join graph such that:

- for every mandatory text element *n* in *t*, there is at least an element *n'* in one of its source parts such that $n \cong n'$;
- for each optional text element *n* in *t*, there may (or not) be an element *n'* in one of its source parts such that $n \cong n'$;
- there is no other partial mapping represented by a graph *G'* included in *G* such that for each element *n* in *t*, if there is an element *n'* in *G* such that $n \cong n'$, then there is also an element *n"* in *G'* such that $n \cong n''$.

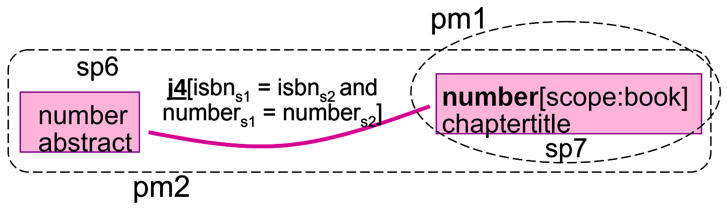


Figure 3-9. Join graph of t3

Consider the subtree *t3* of Figure 3-4, the corresponding join graph is shown in Figure 3-9. It contains two source parts *sp6*, *sp7* and the join *j4*. In this graph, there are two partial mappings: *pm1* containing a single source part *sp7* and *pm2* containing *sp6* and *sp7* related by *j4*. Both are connected acyclic sub-graphs of the join graph and both produce instances for the mandatory text elements $number_{ts}$ and $chaptertitle_{ts}$ in *t3*; *pm1* does not produce instances for the optional element $abstract_{ts}$; *pm2* joins the two source parts; it may produce fewer chapters than *pm1* but more information for every chapter (its abstract).

In the rest of this section, we refer to a partial mapping by the source part name if it contains a single source part, or by the names of the corresponding joins if it contains more than one source part. For example, *pm1* and *pm2* are denoted by {*sp7*} and {*j4*} respectively.

Generating Target Mappings

The mappings for the whole target schema are generated from the partial mappings. The partial mappings of the different target subtrees are first combined to generate ***candidate mappings.*** Then each candidate mapping is checked to see if the parent-child relationships between the target subtrees are satisfied. A candidate mapping satisfying these relationships is a target mapping. The candidate mappings that do not satisfy the parent-child relationships are discarded. The algorithms for generating candidate mappings and checking these mappings are described in Sections 5.1 and 5.2 respectively.

The target mappings are abstract queries that can be translated into another language. In Section 5.3, we present an algorithm for translating abstract target mappings into XQuery.

Algorithm 3-5 describes partial mapping definition. It takes as input one target subtree (*st*) and the corresponding join graph *G(SP, J)* where *SP* represents the set of elements (the source parts) and *J* the set of edges (the candidate joins). The algorithm uses a recursive procedure to browse the join graph and produces the set of all the partial mappings (*PM*) for *st*; each partial mapping in *PM* is represented by the corresponding sub-graph.

6. Generating Target Mappings

The mappings for the whole target schema are generated from the partial mappings. The partial mappings of the different target subtrees are first combined to generate ***candidate mappings***. Then each candidate mapping is checked to see if the parent-child relationships between the target subtrees are satisfied. A candidate mapping satisfying these relationships is a target mapping. The candidate mappings that do not satisfy the parent-child relationships are discarded. The algorithms for generating candidate mappings and checking these mappings are described in Sections 5.1 and 5.2 respectively.

The target mappings are abstract queries that can be translated into another language. In Section 5.3, we present an algorithm for translating abstract target mappings into XQuery.

Algorithm 3-5. Partial mapping definition

```
Partial_Mappings_Definition(G(SP, J), st, PM)
Begin
    PM := ∅;
    for each source part sp in SP:
        J' := ∅;
        SP' := {sp};
        Build_Partial_Mapping (G'(SP', J'), G(SP, J), st, PM);
End

Build_Partial_Mapping (G'(SP', J'), G(SP, J), st, PM)
Begin
    if mandatory_text_elements(SP', st)
        //returns true if SP' contains all the mandatory text elements in st
    then
        PM := PM ∪ {G'(SP', J')};
            //adds a new partial mapping represented by the graph G' to set PM
    for each join j between the source parts sp and sp' such that sp' ∈ SP' and sp ∉ SP'
        SP'' := SP' ∪ {sp};
        J'' := J' ∪ {j};
        if G''(SP'', J'') ∉ PM
            //adding the edge representing the join j to the subgraph G' does'nt give an element of PM
        then
        Build_Partial_Mapping (G''(SP'', J''), G(SP, J), st, PM);
End
```

6.1. Generating Candidate Mappings

Candidate mappings are first generated by combining the partial mappings of the different target subtrees. Given the target schema *TS*, a candidate mapping is defined as follows.

Definition 3-5: Candidate Mapping. A candidate mapping cm for the target schema *TS* is a set of partial mappings such that:

- there is at most one partial mapping for each target subtree in *TS*;
- for each mandatory target subtree *t* having no parent subtree, there is one partial mapping for *t*;
- for each subtree *t*, if there is no partial mapping for *t*, then there is no partial mapping for its child subtrees.
- for each mandatory subtree *t* having the parent subtree *t'*, if there is a partial mapping for *t* then there is also a partial mapping for *t'*.

All the source parts and the candidate joins for our example are shown in Figure 3-8; some of the possible partial mappings for each target subtree are the following: *pm3* = {*j2*} and *pm4* = {*j1, j2*} for *t1*; *pm5* = {*j3*} for *t2*; and *pm2* = {*j4*} for *t3*. Since *t2* is optional and its child *t3* is mandatory, each candidate mapping (denoted *cmi*) either contains no partial mapping for both *t2* and *t3* such as *cm1* = {*pm3*} and *cm2* = {*pm4*}, or contains a partial mapping for both *t2* and *t3* such as *cm3* = {*pm3,* pm5, *pm2*} and *cm4* = {*pm4, pm5, pm2*}.

Checking Candidate Mappings

Target mappings are candidate mappings that satisfy the parent-child relationships between the target subtrees. Consider a target subtree *t,* its parent subtree *t'* and their respective partial mappings *pm* and *pm'; pm* and *pm'* preserve the parent-child relationship between *t* and *t'* if the following conditions hold:

- there is a source part *sp* in *pm* and a source part *sp'* in *pm'* that are in the same source;
- there is either an element in *sp* with respect to which all the elements in *sp'* are monovalued, or an element in *sp'* with respect to which all the elements in *sp* are monovalued.

If there is an element in *sp* with respect to which all the elements in *sp'* are mono-valued, then for every instance of *sp* we can find the corresponding instance of *sp',* and for every instance of *sp'* we can find the corresponding instances of *sp*. The parent-child relationship is therefore satisfied.

Algorithm 3-6 describes candidate mapping generation. This algorithm takes as input the target schema *TS* and the sets of partial mappings *PM1, ..., PMn* corresponding respectively to the subtrees *t1, ..., tn* in *TS*. The algorithm generates the set of candidate mappings *CM* by enumerating all the combinations of partial mappings and checking if each combination satisfies the above definition.

6.2. Checking Candidate Mappings

Target mappings are candidate mappings that satisfy the parent-child relationships between the target subtrees. Consider a target subtree *t*, its parent subtree *t'* and their respective partial mappings *pm* and *pm'*; *pm* and *pm'* preserve the parent-child relationship between *t* and *t'* if the following conditions hold:

- there is a source part *sp* in *pm* and a source part *sp'* in *pm'* that are in the same source;
- there is either an element in *sp* with respect to which all the elements in *sp'* are monovalued, or an element in *sp'* with respect to which all the elements in *sp* are monovalued.

If there is an element in *sp* with respect to which all the elements in *sp'* are mono-valued, then for every instance of *sp* we can find the corresponding instance of *sp'*, and for every instance of *sp'* we can find the corresponding instances of *sp*. The parent-child relationship is therefore satisfied.

Algorithm 3-6. Candidate mapping generation

```
Candidate_Mapping_Generation(TS, PM1, …, PMn, CM)
Begin
    CM := ∅; //each element of CM is a set of partial mappings
    for each target subtree ti in get-mandatory-top-subtrees(TS)
        // returns the subtrees in TS that are mandatory and has not parent subtrees
        if PMi == ∅
        then return (∅);
        //if a mandatory top subtree has no partial mapping, the target schema has no target mapping
        if CM == ∅
        then
            for each partial mappings pm in PMi
                CM := CM ∪ {pm};
        else
            for each set S in CM
                for each partial mapping pm in PMi
                    S' := S ∪ {pm};
                    CM := CM ∪ {S'};
                CM := CM - {S};

    for each subtree ti in TS not in get-mandatory-top-subtrees(TS) from top to down:
        if top(ti) //returns true if ti has not parent subtree
            for each set S in CM
                for each partial mapping pm in PMi
        else
            for each set S  in CM
                if contains_parent_mapping(ti, S)
                    //returns true if the set S contains a partial mapping for the parent subtree of ti
                then
                    for every pm in PMi
                        S' := S ∪ {pm};
                        CM := CM ∪ {S'};
                if mandatory(ti)
                then CM := CM - {S};
End
```

Given a target schema *TS* and a candidate mapping *cm* for *TS*, a target mapping is defined as follows.

Definition 3-6: Target Mapping. A target mapping for a target schema *TS* is a candidate mapping cm such that for each pair of subtrees *t* and *t'* in *TS* such that *t* is the parent of *t'*, if there are two partial mappings *pm* and *pm'* for *t* and *t'* in cm respectively, then *pm* and *pm'* preserve the parent-child relationship between *t* and *t'*.

For the target schema of our example, there are two parent-child relationships to check: one between *t1* and *t2* and the other between *t2* and *t3*. Consider the candidate mapping *cm4* = {*pm4, pm5*, *pm2*}. The parent-child relationship between *t1* and *t2* is satisfied in *cm4* because every element in *sp5* (involved in *pm5*) is monovalued with respect to both $authorid_{s2}$ in *sp3* (involved in *pm4*). The parent-child relationship between *t2* and *t3* is also satisfied because every element in *sp5* is monovalued with respect to both $number_{s2}$ and $chaptertitle_{s2}$ in *sp7* (in *pm2*). Therefore, *cm4* is a target mapping for *TS*.

The candidate mappings *cm1* and *cm2* are also target mappings because both contain a single partial mapping. The candidate mapping *cm3* does not lead to a target mapping because the parent-child relationship between *t1* and *t2* is not satisfied.

At this stage of the generation process, we obtain a set of mappings; each one populates all the mandatory elements of the target schema and satisfies the structure of the target schema. We can characterize mappings as ***sound***, ***complete*** or ***exact*** followings the definition in [Len02]: a sound mapping assures that every instance of the target schema can be found in the source instances; a complete mapping assures that every instance of the sources can be found in the target instances; and an exact mapping is mapping that is at the same time a sound and complete. Our result mappings are **sound *mappings*** according to the definition.

Other target mappings can be derived by applying set-based operations like Union, Intersection and Difference to two or more mappings. For example the union of *cm1* and *cm4* is a new target mapping that takes the union of *pm4* and *pm3* for *t1*, *pm5* for *t2* and *pm2* for *t3*.

Considering the set of generated mappings, the choice of the "best" mapping according to the preferences of a given user cannot be done automatically: the user will have to select one mapping among the proposed ones. We can also take advantage of these alternative mappings. For example, if some data source used in the mapping becomes unavailable, we can choose (if it exists) an alternative mapping that does not use this source. We can also classify the different mappings according to some quality criteria and select an appropriate one for each user according to his preferences [KPS05].

6.3. Translating Target Mappings into XQuery

The target mappings are expressed in an abstract language and they can be translated into a standard query language. In this section we present our algorithm to translate a mapping into XQuery.

Each partial mapping is translated into a `FWR` (`For-Where-Return`) expression [xquery]. The organization of the different `FWR` expressions for a target mapping follows the organization of the target schema. For each target subtree *t* and its parent *t'*, the `FWR` expression of *t* is nested in the `FWR` expression of *t'*. Figure 3-10 shows the translation to XQuery of the target mapping *cm4*.

```
<library>{
    for $au in distinct-values(S1/library/author/id, S1/library/address/authorid,
                                        S2/library/book/authorids),
    for $sp1 in S1/library/author[id = $au][1],
    for $sp2 in S1/library/address,
    for $sp3 in S2/library/book/authorid
    where $sp1/id=$sp2/authorid and $sp1/id=$sp3
    return <author>{
            <id>{data($sp1/id)}</id>,
        <name>{data($sp1/firstname), data{$sp1/lastname}}</name>,
        <address>{data($sp2/authoraddress)}</address>
        for $bk in distinct-values(S2/library/book/isbn, S1/library/author/book/isbn),
        for $sp4 in S1/library/author/book[isbn=$bk][1],
        for $sp5 in S2/library/book
        where $sp4/isbn=$sp5/isbn and $sp5/authorid = $sp3
        return <book>{
            <isbn>{data($sp4/isbn)}</isbn>,
            <booktitle>{data($sp4/title)}</booktitle>,
            <priceineuro>{data($sp5/priceindollar)*0.797}</priceineuro>
            for $ch in distinct-values(S1/library/author/book/chapter/number,
                                            S2/library/book/chapter/number),
            for $sp6 in S1/library/author/book/chapter,
            for $sp7 in S2/library/book/chapter[number = $ch][1]
            where $sp6/number=$sp7/number and $sp7 = $sp5/chapter
            return <chapter>{
                <number>{data($sp6/isbn)}</number>,
                <chaptertitle>{data($sp7/chaptertitle)}</chaptertitle>,
                <abstract>{data($sp6/abstract)}</abstract>
            }</chapter>
        }</book>
    }</author>
}</library>
```

Figure 3-10. An XQuery target mapping

Figure 4-2 gives the general form of the FWR expressions we have considered for our mappings. There are two kinds of for clauses. The first kind of for clause is optional and is used to instantiate a variable with a set of values without duplicates that is shown at the first line in Figure 4-2. Such a clause is added in the mapping for every target element n that is defined as a key; a grouping variable is instantiated with the instances of the source elements such that each source element n' satisfies that (i) *n'* ≅ *n* and (ii) *n'* is monovalued with respect to an element bound by a for clause in the same FWR expression. A distinct-values function is used to eliminate duplicates. For example, in the FWR expression for *t1* in Figure 3-10, there is a for clause to bind a grouping variable *$au* to the distinct instances of id_{s1}, $authorid_{s1}$ and $authorid_{s2}$.

The other kind of for clause is used to bind source parts (at the second line in Figure 4-2). A for clause of this kind is added in the mapping for each source part. It binds a variable to an element *n* with respect to which all the other elements of the source part are monovalued. For example, in the FWR expression for *t1* in Figure 4-2, there are three for clauses for *sp1*, *sp2* and *sp3* respectively. Sometimes there are restrictions in the element path in the for clauses of this kind (e.g. *[id = $au][1]* at line 4 in Figure 3-10). The two restrictions select the first element that satisfies the value with respect to a distinct value defined using the first kind of for clause. It is used to eliminate duplicates in the result with respect to the key element.

The `where` clause contains two kinds of conditions. The first one is to describe candidate joins in partial mappings. A condition is added for every join in a partial mapping: (e.g. *$sp1/id=$sp2/authorid*). The other kind of conditions is added to satisfy the parent-child relationship between the subtrees. For a target subtree *t* and its parent *t'*, if their parent-child relationship is satisfied by two source parts *sp* and *sp'*, a join condition is added in the FWR expressions of *t* (e.g. *$sp5/authorid = $sp3* in the FWR expression for *t2*).

```
 [for         $key_variable in distinct-values(path [, path…] )]
 for          $src_part_variable in path [, for $src_part_variable in path …]
 [where       [src_part_join_condition [and src_part_join_condition ..]]
               [and parent_child_condition]
 return       target_subtree_assignments
```

Figure 3-11. General syntax of the For-Where-Return expression

The `return` clause of each `FWR` expression specifies how to populate the corresponding target subtree. The specification corresponds to the structure of this subtree. For every text element of the target subtree, an assignment of a source element is added in the `return` clause.

Erreur ! Référence non valide pour un signet. describes the target mapping translation to XQuery. It takes as input a target schema *TS* and a target mapping *M* for *TS*. The algorithm produces the translation *XM* of *M* in XQuery. The principle of the algorithm is to translate for every target subtree in *TS* from top to bottom the corresponding partial mapping of *M* into a `FWR` expression. If this subtree has a parent subtree, the algorithm also adds a join condition in the `FWR` expression to represent the parent-child relationship.

Algorithm 3-7. Target mapping translation into XQuery

```
XQuery_Translation(TS, M, XM)
Begin
    XM := null;
    if root_not_in_subtree(TS) //return true if the root of TS is not in a target subtree
    then
        add_tags(TS, XM);
            //add the tags of the XML elements of TS that are not in any target subtree
    for each subtree t in TS from top to down
        pm := get_partial_mapping(t,M) //returns the partial mapping for t in M
        if pm ≠ null
        then
            xq_pm := translate_FWR(pm); //return the translation of pm in XQuery
            t' := get_parent_subtree(t); //get the parent subtree of t
            if t' ≠ null
            then add_where_condition(t, t', M, xq_pm);
                //get the pair of source parts satisfying the parent child relation between t and t' in M
                //and add it in exp
            include(xq_pm, XM);
End
```

7. Generating Incomplete Results

The mapping generation approach presented in the previous sections generates a set of mappings that derive instances for all the mandatory components of the target schema. If some mandatory components are not found in the sources or the parent-child relationships between target subtrees are not satisfied by all the candidate mappings, no target mapping is returned, which means that there is no mapping meeting the users expectations described in the target schema in terms of components or in terms of structure.

If no mapping is found for the target schema, one possible solution is to propose a new target schema that can be satisfied by the sources and that is as close as possible to the initial target schema. For this purpose, we extend our approach by relaxing some cardinality or structural constraints of the original target schema such that a mapping can be found for the new schema from the sources. This process is performed only if no mapping is found for the initial target schema, and the users will decide whether to accept or not the resulting target schema and the corresponding mappings based on their specific needs.

There are three cases in which no mapping is found for the target schema:

- there is no corresponding source element for some mandatory text elements of a given target subtree;
- all the mandatory text elements have corresponding elements in the sources, but some joins are missing to relate these source elements;
- all the target subtrees have partial mappings, but the parent-child relationship between two target subtrees is not satisfied.

In the first two cases, no partial mapping can be found for the target subtree. For the third case, no candidate mapping satisfies the structural links of the target schema. In the rest of this section, we present the adaptation of our approach to propose a new target schema and the corresponding mappings in all these situations. The target schema can be modified in two ways: changing some mandatory elements into optional or restructuring the target schema. We also present the global process for generating incomplete results.

7.1. Changing Mandatory Elements to Optional

For a given target subtree *t*, the partial mapping is defined as a connected acyclic subgraph of the join graph that contains an equivalent element for each mandatory element of *t*. If no connected acyclic subgraph meets this condition, then no partial mapping will be found.

Consider the case in which there is no corresponding source element for some mandatory text elements of the target subtree. Our adaptation in this case consists in modifying the target schema by changing the mandatory cardinality of this element into optional. A partial mapping can then be generated from the subgraph for the modified target subtree.

Consider the example in Figure 3-12 (a). There is no partial mapping for the target subtree of *TS1* from the source schema *S3* because there is no correspondence for the mandatory element *tel*. Consider a new target schema *TS2* which is the same as *TS1* except that the cardinality of *tel* is changed to optional (shown in Figure 3-12 (b)), {*j1*} is a partial mapping for the subtree of the *TS2* that derives instances for all the other text elements of *TS2* except *tel*.

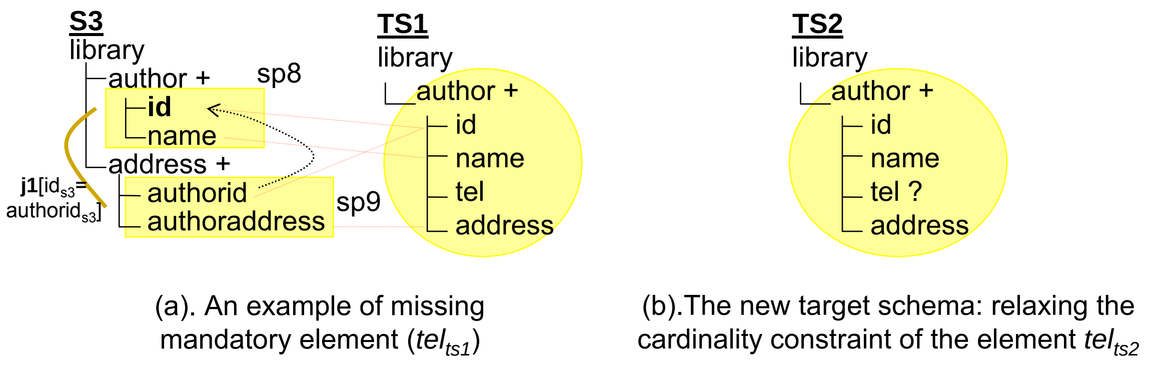

S3
library
author +
id
name
address +
authorid
authoraddress
sp8
sp9
j1[id$_{s3}$=authorid$_{s3}$]
TS1
library
author +
id
name
tel
address
TS2
library
author +
id
name
tel ?
address
(a). An example of missing mandatory element (tel_{ts1})
(b).The new target schema: relaxing the cardinality constraint of the element tel_{ts2}

Figure 3-12. Changing mandatory elements to optional

Another situation in which there is no partial mapping for a target subtree is when all the mandatory elements of the target subtree have equivalent elements in the join graph, but there is no connected subgraph containing all these mandatory elements; these elements are in different connected subgraphs. In this case, the system interacts with the user to get his or her preferences and allows the selection of one of the subgraphs. A new target schema is produced from the initial one by changing to optional the cardinality of all the mandatory elements that are missing in the subgraph selected by the user. An alternative solution is to choose the connected subgraph containing the maximum number of mandatory elements.

After relaxing the cardinality constraint of a target element, some steps of mapping generation are re-executed: the partial mapping definition for the considered target subtree, the candidate mapping generation and the parent-child relationship checking. This is shown in Figure 3-13.

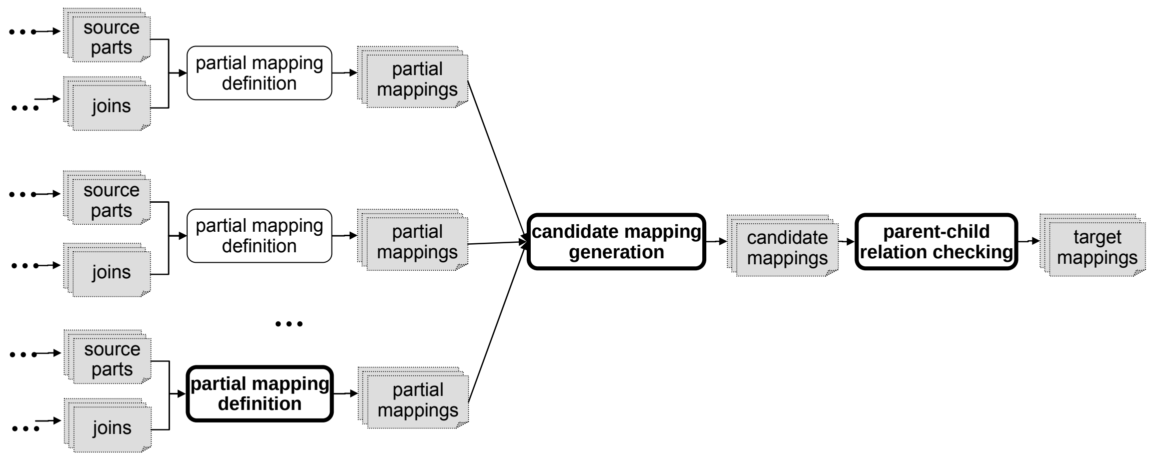

source parts
joins
partial mapping definition
partial mappings
...
candidate mapping generation
candidate mappings
parent-child relation checking
target mappings

Figure 3-13. Re-execution of the mapping generation tasks after the relaxation of a cardinality constraint

7.2. Restructuring the Target Schema

In our generation process, a candidate mapping leads to a target mapping only if all the parent-child relationships between the different target subtrees are satisfied. If there is no target mapping because these relationships are not satisfied, our adaptation consists in proposing a new target schema by relaxing some structural constraints. Consider a subtree *t* and one of its child subtree *t'*. If a candidate mapping cannot satisfy the parent-child relationship between *t* and *t'*, the target schema can be restructured by removing the link between *t* and *t'* and adding a link between *t'* and the parent subtree *p* of *t*. This is done by removing the edge between the root element of *t* and its parent and adding an edge between the root element of *t* and the root element of *p*. If the parent-child relationship between *t'* and the parent of *t* is not satisfied, the process of restructuring is repeated until a subtree is found for which the parent-child relationship with *t'* is satisfied. If no subtree satisfying this condition is found, then the target schema is restructured as follows:

- if the root *r* of the target schema is not in a subtree, then *t'* is related to *r*;
- if the root *r* of the target schema belongs to a subtree, then a new root element *r'* is created and two links are added: one between *r'* and *r* and the other between *r'* and *t'*.

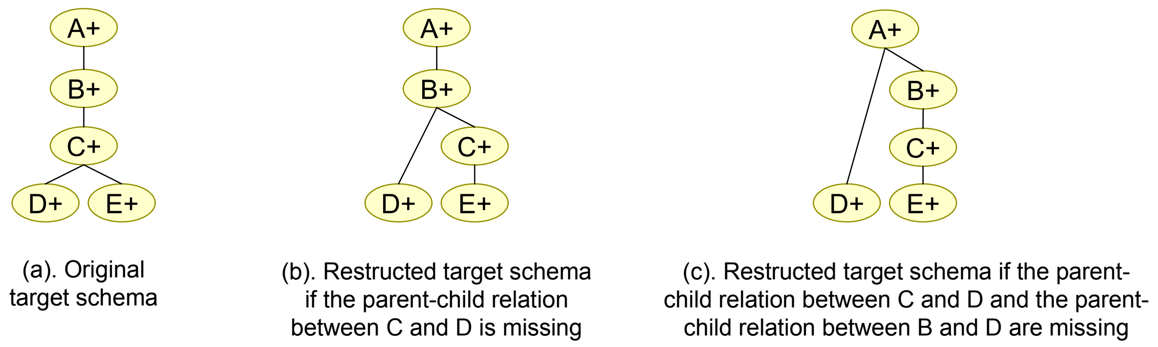

A+
B+
C+
D+
E+
(a). Original target schema
A+
B+
C+
D+
E+
(b). Restructed target schema if the parent-child relation between C and D is missing
A+
B+
C+
D+
E+
(c). Restructed target schema if the parent-child relation between C and D and the parent-child relation between B and D are missing

Figure 3-14. Restructuring the target schema

Consider the target schema of Figure 3-14 (a); if the parent-child relationship between *C* and *D* is not satisfied, then a new schema is proposed as shown in Figure 3-14 (b): the link between *C* and *D* is replaced by a link between *B* and *D*. If this new link is not satisfied, the schema is restructured as shown in Figure 3-14 (c).

After restructuring the target schema, the candidate mappings for the given schemas need to be re-checked to make sure they satisfy the remaining parent-child relationships of the target schema. This is illustrated in Figure 3-15.

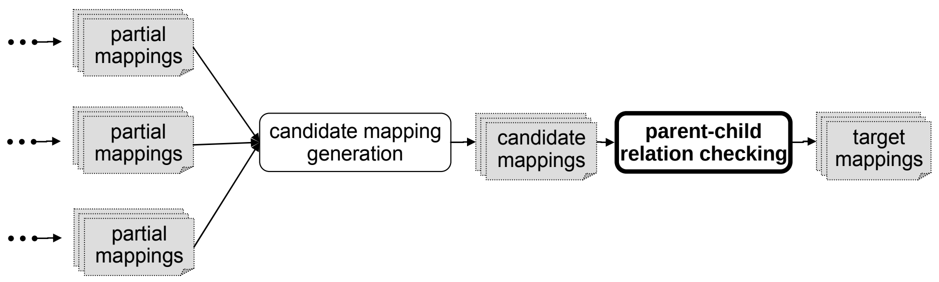


Figure 3-15. Re-checking parent-child relationships after the relaxation of a structural constraint

7.3. Global Process for Generating Incomplete Results

The process of generating incomplete results is interactive and the user decides the most suitable relaxation. He can first decide to accept or not to have incomplete results. The user can also decide to keep one incomplete target schema rather than another if several solutions are possible. There are three situations in which interaction with the user is required:

– there are several cardinality relaxations that lead to a mapping;
– there are several restructuring that lead to a mapping;
– both restructuring and cardinality relaxation lead to a mapping.

Consider the first case that the user decides to relax the cardinality constraint of different elements. It may happen that no partial mapping is generated for the join graph of a target subtree; suppose that several connected subgraphs miss equivalent elements for different target elements. The user can decide to relax the cardinality constraint of one of the target element giving preference to one or several subgraphs rather than the others.

Consider two candidate mappings deriving instances for the same target subtrees. Both of them do not satisfy all the required parent-child relationships, but each candidate mapping satisfies different ones. The user can choose one of the two candidate mappings because the missing parent-child relationships in this one are less important than the ones in the other.

Users can also choose between relaxing some cardinality constraints and restructuring the target schema. Consider the example in Figure 3-16. Given the target schema *TS1* and three source schemas *S4*, *S5* and *S6*, there is no mapping that can be generated from the sources to satisfy the target schema. The target schema contains two subtrees *t1* and *t2* and each one has two source parts in the sources: *sp1* and *sp2* for *t1*, and *sp3* and *sp4* for *t2*. No join can be found between the source parts because there is neither a key nor a reference in the sources. One partial mapping {*sp1*} is defined for *t1* and two partial mappings {*sp3*} and {*sp4*} are defined for *t2*. Such partial mappings form two candidate mappings but none of them can satisfy the parent-child relation between *t1* and *t2*. In this case, there are two possibilities to generate incomplete results. The first one is to restructure the target schema such that *t2* becomes the child of the root *library*$_{ts}$. This relaxation allows producing two target mappings from the two candidate mappings. There is another option of relaxation: changing the cardinality constraint of the element *affiliation*$_{ts}$ to optional. Such relaxation leads to another partial mapping {*sp2*} for *t1*. A candidate mapping can be generated from the combination of {*sp2*} and {*sp4*}; it satisfies the parent-child relationship and leads to a target mapping.

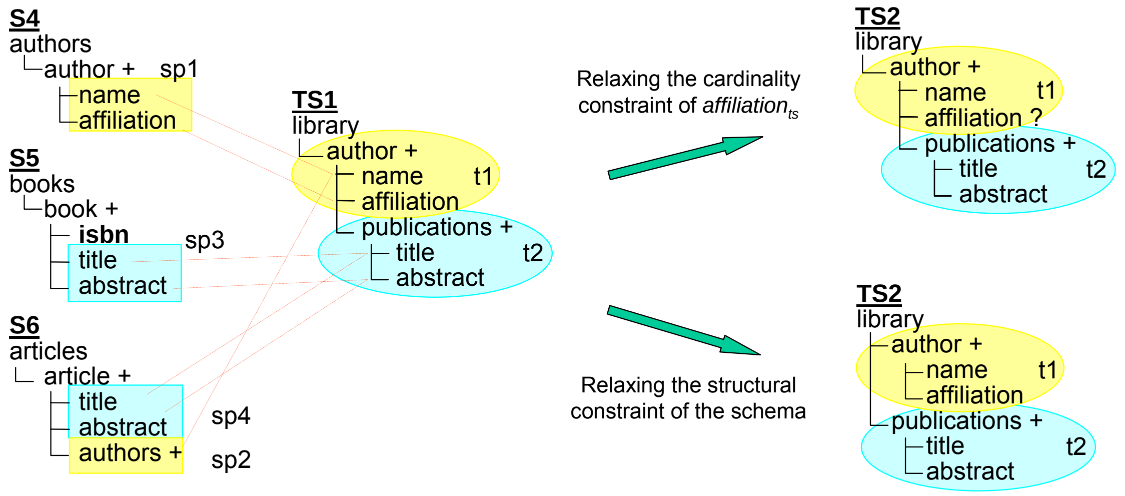


Figure 3-16. An example of multiple possibilities of target constraint relaxation

The system generates incomplete result and provides the facilities to allow users to explore the intermediate results, especially the join graphs and the candidate mappings. It can also make some suggestions of incomplete results. However, only users can decide if they accept them or not and to select one among several possibilities. Figure 3-17 shows the global process of incomplete mapping generation. All the steps presented in bold require user interaction. If there is no candidate mapping for the target schema, then there is at least one mandatory target element that is not involved in a correspondence and the system needs to relax cardinality constraints of such target elements. If there is a candidate mapping, then the user can choose to relax a structural constraint or a cardinality constraint. The user also decides to continue the relaxation if the new target schema is satisfactory or if there is still no mapping generated for the new schema.

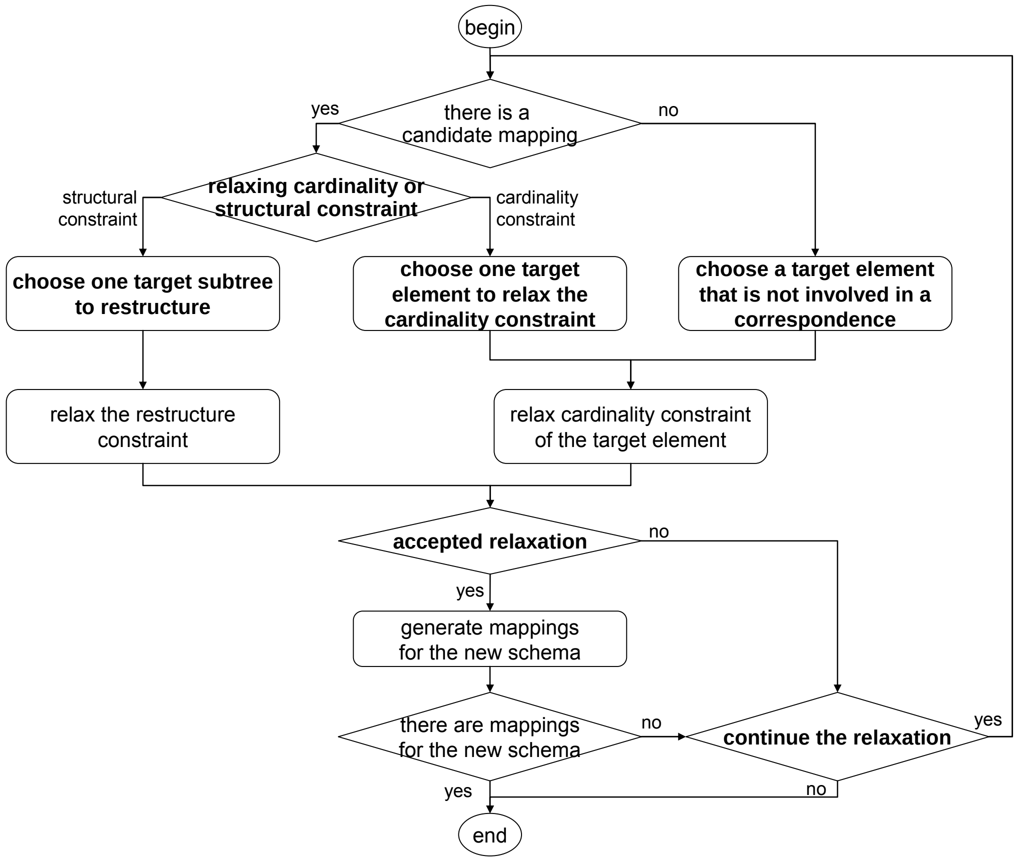

begin
there is a candidate mapping
yes
no
relaxing cardinality or structural constraint
structural constraint
cardinality constraint
choose one target subtree to restructure
choose one target element to relax the cardinality constraint
choose a target element that is not involved in a correspondence
relax the restructure constraint
relax cardinality constraint of the target element
accepted relaxation
no
yes
generate mappings for the new schema
there are mappings for the new schema
no
continue the relaxation
yes
yes
no
end

Figure 3-17. The global process for generating incomplete results

8. Conclusions

In this chapter, we have presented our approach on automatic mapping generation and its adaptation for generating incomplete results. The approach takes one target schema and several source schemas, all described in XML Schema as input and it generates mappings for the target schema from the source schemas using a set of semantic correspondences. The mapping generation approach creates a set of mappings such that each mapping represents alternative ways to derive instances of the target schema. If the data sources do not allow generating a mapping satisfying the users' needs as represented by the target schema, new target schemas are proposed by relaxing some cardinality constraints or some structural constraints on the target schema such that a mapping can be found.

The generated mappings are sound mappings according to the definition of [Len02]. They derive instances that conform to the structure and the cardinality constraints of the target schema. Semantic correspondences allow finding relevant sources for the target and the source constraints allow relating objects. The result mappings therefore depend on the input metadata: if the set of available metadata is complete, more possibilities are found to generate mappings that conform to the user's expectation.

Compared to the existing approaches, our approach generates mappings from multiple source schemas and it generates mappings involving inter-source operations. It does not assume that the target schema and the source schemas have homogeneous structures. It generates mappings expressed in an abstract format and they can be translated into a specific query language; we have presented the translation of the abstract mappings into the standard language XQuery.

Our approach enumerates all the possible solutions to generate mappings and produces a set of mappings that represent different ways to define the target from the sources. In case of a large number of source schemas, such method presents two drawbacks: (i) the time needed for enumerating all the possible combinations between the sources may be very large; (ii) many mappings may be generated which makes the choice of the "best" ones difficult. One of our perspectives is therefore to use some methods for data quality evaluation. It can be used to automatically classify the result mappings. It can be also used inside of the mapping generation process to eliminate some intermediate results that will give mappings of unsatisfying qualities. We have proposed a platform as a first attempt for evaluating data quality of mappings generated using our approach [KPS05].

Another perspective consists in adapting this mapping generation approach for peer-to-peer systems, in which there is no global view of the peers. Every peer has only information about his neighbor. A peer sends a query to his neighbors; either one neighbor answers the request or he forwards the request to his own neighbors and so on till one peer answers the request. In this context, mappings can be built dynamically. Some partial results may be first found from the neighbors of the target and then they will be extended with the information provided by the other peers to produce the final mappings. A cost model may be introduced and some optimization algorithm may be used to limit the full exploration of the peers.

Chapter 4. Automating Mapping Adaptation when Schemas Evolve

1. Introduction

Mappings are defined between a target schema and a set of source schemas to express the way instances of the target schema are derived from the instances of the sources. In highly dynamic environments with no centralized authority such as the Web, sources may evolve without prior notification. This evolution may concern not only their content but also their schemas and their query capabilities. When this happens, mappings that depend on these schemas may become invalid or inconsistent and they have to be adapted to conform to the new schema structures and semantics.

If the number of the source schemas is small and if the schemas themselves are simple, manually browsing a short list of mappings to perform the required changes is a feasible option, but as the number of the schemas increases and their structure becomes more complex, the effort needed to perform this task is considerable, since it requires rewriting of large complex transformation queries and programs. If some change occurs in the sources or the target schema, one solution is to restart the generation process; but this can be costly, especially when the changes have little impact on the mappings. For example, the renaming of a source element can be propagated in a very simple way in the existing mappings and does not require generating the mapping from scratch.

Some approaches [BFK03, LNR02, MP02, VMP04 and YP05] have been proposed to adapt mappings automatically when the underlying source schemas evolve. Among the existing works, BFK03 and LNR02] consider relational schemas and mappings expressed in SQL; [VMP04 and YP05] consider XML schemas and mappings expressed in a specific format; [MP02] considers schemas expressed using an Entity-Relationship like model with mappings expressed using a serial of transformation operations. All these approaches assume that semantic correspondences are provided. Three of the proposed approaches [BFK03, LNR02 andVMP04] are incremental approach. Each of these approaches considers a set of schema changes and adapts the mapping for every change. The approaches proposed in [MP02 and YP05] are mapping composition approaches. They use a mapping to express the evolution between an original schema and a new schema; the mapping adaptation consists in composing the original mapping with the mapping describing the evolution to get an adapted mappings. Two of the proposed approaches [BFK03 and VMP04] can be considered as an incremental execution of an existing mapping generation approach. These approaches consider that all the intermediate results of the mapping generation process are provided and, for every change, they re-compute step-by-step the intermediate results and the final results of the mapping generation. The approaches presented in [MP02 and YP05] are mapping composition approaches; they consider a specific mapping generation approach to produce the mappings describing the schema evolutions. The approach presented in [LNR02] adapts mappings for source changes. This adaptation mainly consists in searching for substitutions in case of removal of source attributes or source relations. The adaptation is performed using some available metadata about the sources such as the possible joins existing between source relations and the inclusion between sets of instances of different sources, etc.

One limitation of the existing approaches for mapping adaptation is that most of them rely on a specific mapping generation approach to either provide all the intermediate results of the mapping generation or generate the mappings after some changes occurring in some schema. The work presented in [LNR02] does not make such assumption but it does not support target evolution and requires a large volume of source descriptions that are difficult to be obtained in practical situations, such as the set of possible joins between the sources.

Our goal is to propose an approach to enable the adaptation of the mappings after changes occurring in the schemas without making any assumption about the methodology used to generate these mappings: they can be either generated by some mapping generation tool or even manually specified by a designer. Our adaptation approach supports both source schema changes and target schema changes, representing the evolution of the users' needs. We consider schemas described in XML Schema [xsd] and mappings expressed using XQuery [xquery]. Our proposal is an incremental approach and it adapts the mapping for every considered change in three steps:

- The original mapping is first checked to see if it is ***affected*** by the change. There are two situations in which a mapping can be affected: either the target schema has changed, or some source elements that are used in the original mapping have been moved or removed.
- If the original mapping is affected, it is checked to see if it is ***adaptable***, which means that there is at least one solution to adapt it to the new schemas. If some new components have been added in the target schema, the mapping needs to be extended to allow deriving instances for these new components. If some source components used in the mappings are moved or removed, some substitutions needs to be found to replace them. For a given change, all the possible ways to adapt the mapping are proposed.
- If the mapping is adaptable, ***adapted mappings*** are generated using all the adaptation solutions found at the second step representing alternative ways to conform to the new schemas.

In our adaptation approach, we consider a set of changes and we have defined specific adaptation algorithms for every change type. As examples, adapting a mapping after a source element removal consists in finding the substitutions of the removed element, and adapting a mapping after a target element removal consists in removing the corresponding assignment from the mapping.

This chapter is organized as follows. Section 2 presents the basic assumptions for mapping adaptation, and especially concerning the schemas representation and the metadata used in our approach. Then Section 3 presents the general form of the mappings considered in our approach. Section 4 gives an overview of the mapping adaptation approach. Sections 5 and 6 present the detailed algorithms of mapping adaptation for source evolution and for target evolution respectively. Section 7 concludes the chapter.

2. Basic Assumptions

As in our proposal for mapping generation, we consider that both the target schema and the source schemas are provided and they are expressed using XML Schema [xsd]. Recall that there exist two languages to describe XML documents: DTD (Document Type Declaration [dtd]) and XML Schema [xsd]. XML Schema includes the full capabilities of DTDs and it can also express additional information such as relative keys. We have considered that all the schemas are described using XML Schema. We also consider that some metadata is available about the description of the target and the source schemas and also about the semantic links between the sources and the target schema; these links are represented by a set of semantic correspondences.

Similarly to the mapping generation approach presented in Chapter 3, the quality of the result of the mapping adaptation process will depend on the quality of the metadata. For example, if an element is removed from the sources and if the set of correspondences provided to the system is complete, the system can identify more substitutions for the removed elements.

In the rest of the section, we first present our assumptions about the schema representation and the semantic correspondences in Sections 2.1 and 2.2 respectively.

2.1. Schemas

As for the automatic mapping generation approach presented in Chapter 3, we consider in our mapping adaptation approach that both the target schemas and the source schemas are described using XML Schema [xsd] and we assume the a set of semantic correspondences between the target schema and the source schemas is provided.

The presentation of the main features of XML Schema is presented in Section 2.1 of Chapter 3. Figure 4-1 shows an example of two source schemas and a target schema in XML Schema. Every element in a XML Schema is either a text element (e.g. $authorId_{s1}$), or an internal element (e.g. $chapter_{s1}$). Elements are suffixed by the name of their schema.

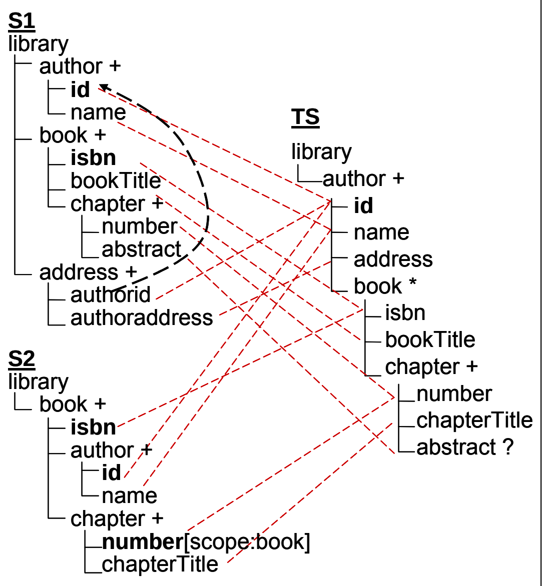
Based on the attributes `minOccurs` and `maxOccurs`, each element is monovalued (`maxOccurs` = 1) or multivalued (`maxOccurs` > 1) and it is optional (`minOccurs` = 0) or mandatory (`minOccurs` > 0). In Figure 4-1, the symbol '+' represents a multivalued and mandatory element (e.g. $book_{s2}$); '*' represents a multivalued and optional element (e.g. $book_{ts}$); and '?' represents a monovalued and optional element (e.g. $abstract_{ts}$). An element without symbol is monovalued and mandatory (e.g. id_{s1}).

Key and referential constraints can be expressed in schemas. Keys are defined on mandatory elements and there are two kinds of keys: absolute Keys and relative keys. Absolute keys are defined for the whole schema. Relative keys are defined for a portion of the schema and its scope is marked by an antecessor of the identified element, except the root. In Figure 4-1, the elements written in bold represent keys. In case of a relative key, the name of the key element is followed by a bracket in which there is the scope of key (e.g. $Number_{s2}$ is a relative key and its scope is $Book_{s2}$). Every referential constraint is a set of text elements referencing another set of text elements defined as a key. They are represented by arrows in Figure 4-1 (e.g., $AuthorId_{s1}$ references Id_{s1}). In our mapping adaptation, we consider key and referential constraints in the source schemas. In the target schema, we consider only key constraints but not referential constraints.

2.2. Semantic Correspondences

As for the mapping generation approach, we consider that a set of semantic correspondences between the target schema and the source schemas is provided. At this stage of the work, only 1-1 correspondences are supported; recall that a 1-1 correspondence between two elements *n* and *n'* states that these two elements represent the same concept and it is denoted $n \cong n'$. In Figure 4-1, dotted lines represent correspondences. Semantic correspondences between elements of different sources can be derived from the correspondences between the target elements and the source elements.

The same notation is used to represent correspondences between sets of elements. There is a correspondence between two sets of elements *s1* and *s2* if (i) both *s1* and *s2* contain the same number of elements (ii) and for each element n1 in s1 there is exactly one element *n2* in *s2* such that $n1 \cong n2$, and vice versa. The correspondence between the two sets *s1* and *s2* is denoted $s1 \cong s2$ (e.g. $\{isbn_{s1}, bookTitle_{s1}\} \cong \{isbn_{ts}, bookTitle_{ts}\}$).



```
<library>{
    for $ka in distinct-values(S1/library/author/id,
            S1/library/address/authorid, S2/library/book/author/id)
    for $v1 in S1/library/author[id=$ka][1],
    for $v2 in S1/library/address,
    for $v3 in S2/library/book/author
    where $v1/id=$v2/authorid and $v1/id=$v3/id
    return <author>{
        <id>{data($v1/id)}</id>,
        <name>{data($v1/name)}</name>,
        <address>{data($v2/authoraddress)}</address>
        for $v4 in S1/library/author/book,
        for $v5 in S2/library/book
        where $v4/isbn = $v5/isbn and $v3 = $v5/author
        return <book>{
            <isbn>{data($v4/isbn)}</isbn>,
            <bookTitle>{data($v4/title)}</bookTitle>
            for $v6 in S1/library/author/book/chapter,
            for $v7 in S2/library/book/chapter
            where $v6/number = $v7/number and $v7 = $v5/chapter
            return <chapter>{
                <number>{data($v6/isbn)}</number>,
                <chapterTitle>{data($v7/chapterTitle)}</chapterTitle>,
                <abstract>{data($v6/abstract)}</abstract>
            }</chapter>
        }</book>
    }</author>
}</library>
```

Figure 4-1. Schemas, semantic correspondences, and mapping

3. Mapping Representation

Mappings are defined between the target schema and the source schemas to specify how instances of the target schema can be derived from the instances of the sources. They can be generated automatically by some generation tools [KX05, PVM02] or expressed manually by users. We consider that the mappings used in our approach are expressed in XQuery [xquery]. XQuery is the standard query languages for XML sources and it is largely used in today's applications involving XML sources. In our work, we consider that the mappings are expressed following a specific XQuery pattern that we have defined. In XQuery, many different expressions can represent the same meaning. The goal of the proposed XQuery pattern is to restrict the description of a mapping to a limited number of XQuery clauses. This allows simplifying the query syntax. Mappings that use clauses that are not in our pattern have first to be translated into our specific representation. Appendix II shows the translations of some typical mappings into our specific format.

Before defining the general form of the mappings supported by our approach, we present the definition of a `For-Where-Return` expression (`FWR` expression for short):

Definition 4-1. `FWR` (`For-Where-Return`) expression. A `FWR` expression contains:

- zero to several `for` clauses to define variables that will be used in this expression;
- possibly one `where` clause with several conditions that the variables defined by the `for` clauses have to satisfy;
- a `return` clause that contains assignments for target elements; all the source elements used for an assignment are related to the variable defined by a `for` clause of the same expression; the `return` keyword can be omitted if there is no `for` clause in the same expression.

Figure 4-2 gives the general form of a `FWR` expression. The `for` clauses are used to define variables and to bind them to source elements. There are two kinds of `for` clauses: the ones that bind a variable to a source element; and the one that associates with a variable with a set of values without duplicates.

```
[for          $duplicate_elimination_variable in distinct-values(path [, path ..] )]
[for          $element_binding_variable in path [duplicate_elimination_statement] ..]
[where        [join_condition [and join_condition ..]]
               [grouping_condition]]
[return]      target _assignments
```

Figure 4-2. General form of the `For-Where-Return` expressions

The syntax of the `for` clauses that associate a variable to a set of values without duplicates is shown at the first line of Figure 4-2. It associates a variable with the union of instances of some source elements [xquery] and a `distinct-values` function is applied to these source elements to eliminate duplicates in the returned set of instances. We denote this kind of `for` clauses ***duplicate-elimination* `for` *clauses***; variables bound by a duplicate-elimination `for` clauses are denoted ***duplicate-elimination variables***. There is a duplicate-elimination `for` clause at the second line in the mapping of Figure 4-1. It associates the variable *$ka* to a set of author id without duplicates.

The other kind of `for` clauses binds variables to an element of a source schema. This kind of clause is shown at the second line of Figure 4-2. They are denoted ***element-binding* `for` *clauses*** and the corresponding variables are denoted ***element-binding variables***. In this kind of `for` clause, the path of the element to which the clause binds the variable may contain a sequence of two predicates: the first one is a predicate involving a duplicate-elimination variable and the second predicate is *[1]*. The `for` clause at the fourth line of the mapping in Figure 4-1 belong to this kind. It binds the variable *v1* to the source element $author_{s1}$. There are two predicates *[id = $ka][1]* in the path of the element to which the `for` clause binds the variable *$v1*. The first predicate *[id = $ka]* involves the duplicate-elimination variable *$ka* to group the instances of $author_{s1}$ such that their child id_{s1} all have the same value. The second predicate *[1]* selects the first one from the group. We denote the sequence of these two predicates a ***duplicate-elimination statement***. We assume that every duplicate-elimination `for` clause is defined for a key of the target schema to select the unique instances from the equivalent source elements. Duplicate-elimination statements are then used to ensure that the returned instances do not contain duplicates with respect to the values of the key element. For example, the duplicate-elimination `for` clause defining *$ka* and the duplicate-elimination statement *[id = $ka][1]* ensures that the returned instances do not contains duplicates for the target element id_{ts} that is defined as a key.

Two kinds of conditions can be defined in the `where` clause. We call the first one ***join conditions***. Every join condition involves two different source elements that are respectively related to two element-binding variables (e.g. $id_{s1} = authorid_{s1}$) defined in the same `FWR` expression. The second kind of conditions involves the same element related to two different element-binding variables (e.g. *[$v3 = $v5/author]* in the mapping of Figure 4-1). In this case, one of the two element-binding variables belongs to the same `FWR` expression as the condition, and the other is in the `FWR` expression in which the current expression is nested. Using this join condition allow grouping the instances derived by the current expression with respect to the instances derived by the expression in which it is nested. We denote this kind of conditions ***grouping conditions***.

The `return` clause of each `FWR` expression specifies the target assignments. The source element used in every target assignment is related to an element-binding `for` clause of the same `FWR` expression. The target assignments in the `return` clause have the same organization as the assigned elements in the target schema.

Having defined the general form of FWR expressions that we consider, we now define the mappings as follows:

Definition 4-2. Mapping. Given a target schema, a mapping for the target schema consists in a set of `FWR` expressions such that:

- every `FWR` expression derives instances for a subtree of the target schema such that the root of the subtree is multivalued and all the other elements are monovalued;
- for every element *n* and its parent *n'* in the target schema, the `FWR` expression containing an assignment for *n* is nested in the `FWR` expression containing an assignment for *n'*.

Following the definition, every `FWR` expression derives instances for a set of elements where each element is monovalued with respect to the others. Recall that given two elements *n* and *n'* in a schema and their first common antecessor *m*, *n* is monovalued with respect to *n'* if every element on the path from *m* to *n* (except *m*) is monovalued. (ref. Definition 3-2).. The relationship between one element and its multivalued children is represented in the mapping by the relationships between the corresponding `FWR` expressions through a grouping condition. For every element-binding `for` clause that binds a variable *v* to an element *n*, all the source elements that are in a target assignment and that are related to *v* are monovalued with respect to *n*; and *n* is also monovalued with respect to them. Consider again in Figure 4-1 the `for` clause that binds a variable *$v1* to $author_{s1}$. There is a target assignment involving $name_{s1}$: $author_{s1}$ is monovalued with respect to $name_{s1}$ and vice versa. Appendix II shows the transformation of some typical mappings into our representation.

4. An Overview of our Automatic Mapping Adaptation Approach

This section describes the general process of automatic mapping adaptation. As an incremental approach, our system considers a set of changes that can occur in the target schemas or in the source schemas and we propose a specific mapping adaptation procedure for each of these changes.

In our approach, we have considered the following changes both in the target and the source schemas: (i) adding an element, (ii) removing an element, (iii) moving a subtree in the same schema, (iv) renaming an element, (iiv) adding a key definition and (iiiv) removing a key definition. In the source schemas, we also consider the addition and removal of a referential constraint.

The mapping adaptation process is defined independently for each of these changes. Other evolutions can be expressed as a set of changes, possibly partially or totally ordered. As an example, removing a source schema can be represented by the following sequence of changes: first removing all the constraints defined in the schema, and then removing all the elements of the schema in a bottom-up order.

Consider a mapping that is defined for a target schema and some source schemas. For every change occurring in the schemas, we first have to check if the mapping needs to be adapted. If that is the case, the system checks if the mapping can be adapted and produces the different adapted mappings. The general process of mapping adaptation is illustrated in Figure 4-3.

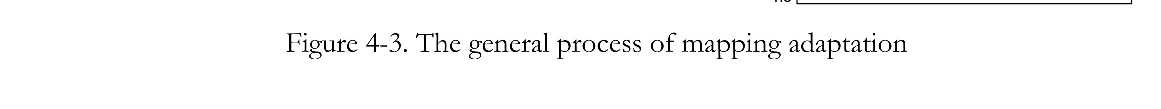
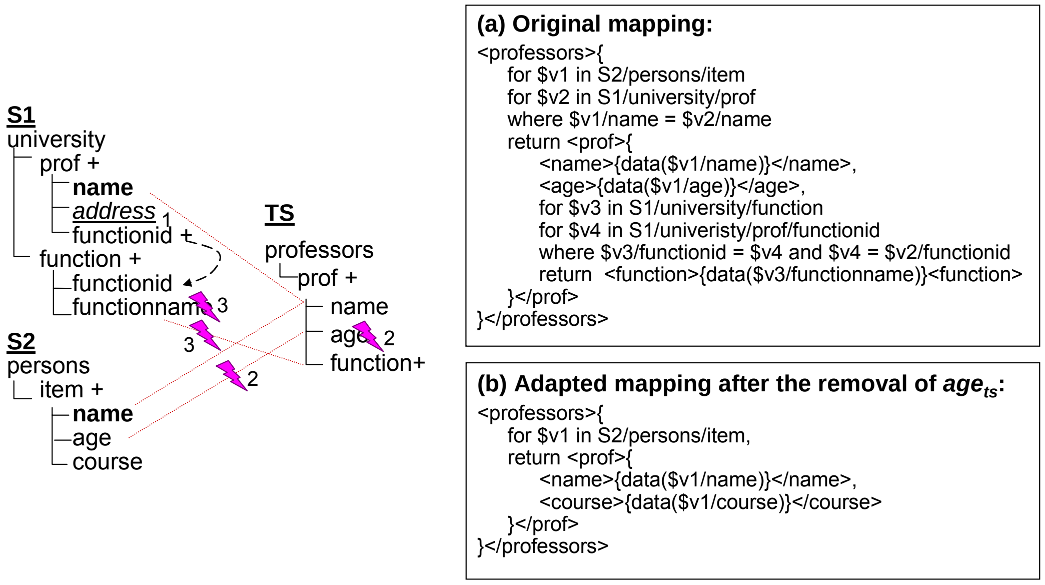

begin
change
Checking if the original mapping is affected
mapping affected
no
yes
Checking if the affected mapping is adaptable
mapping adaptable
yes
no
Adapting the mapping
end

Figure 4-3. The general process of mapping adaptation

A mapping is adapted after a change only if the change concerns an element of the schema that is involved in the mapping. For example if a source is removed from the environment, the mapping is adapted only if it involves this source.

Consider the example shown in Figure 4-4. There is one target schema *TS*, two source schemas *S1* and *S2*, and a mapping defining the target schema *TS* from the source *S2* in Figure 4-4 (a). We consider that three changes occurred in the schemas: (i) adding the element $address_{s1}$ in the source S1, (ii) removing the element age_{ts} from the target schema, (iii) removing the element $functionname_{s1}$ from the source schema S2. The first change does not require the adaptation of the mapping. This change adds a source element and this does not make the mapping obsolete. The mapping needs to be adapted for the second and the third changes. After the removal of the element age_{ts} from the target schema, the original mapping becomes obsolete because the returned instances using the mapping do not conform to the new structure of the target schema. After the removal of the element $functionname_{s1}$ from S2, the mapping also becomes obsolete and an equivalent element has to be found from the sources.



(a) Original mapping:

```
<professors>{
    for $v1 in S2/persons/item
    for $v2 in S1/university/prof
    where $v1/name = $v2/name
    return <prof>{
        <name>{data($v1/name)}</name>,
        <age>{data($v1/age)}</age>,
        for $v3 in S1/university/function
        for $v4 in S1/univeristy/prof/functionid
        where $v3/functionid = $v4 and $v4 = $v2/functionid
        return  <function>{data($v3/functionname)}<function>
    }</prof>
}</professors>
```

(b) Adapted mapping after the removal of *age_{ts}*:

```
<professors>{
    for $v1 in S2/persons/item,
    return <prof>{
        <name>{data($v1/name)}</name>,
        <course>{data($v1/course)}</course>
    }</prof>
}</professors>
```

Figure 4-4. An example of mapping adaptation for three changes

We introduce the notion of ***affected mappings*** to qualify mappings that need to be adapted after a change:

Definition 4-3. Affected mappings. Consider a mapping defined for a target schema from a set of source schemas and consider a change occurring in one of these schemas. The mapping is ***affected*** after the change if the instances returned using this mapping do not conform either to the structure or the semantics of the target schema. Otherwise, the mapping is not affected.

It is not always possible to adapt an affected mapping after a change. If the change adds more requirements to the target schema, then the mapping can be adapted only if the new requirements can be satisfied from the sources. If the change consists in removing some components from the sources, then the mapping can be adapted only if some substitutions can be found to replace this components.

Consider the example of Figure 4-4. There are the two changes that make the original mapping affected: one removes the target element age_{ts}, and the other removes the source element $functionname_{s1}$. After the removal of the element age_{ts}, the mapping can be adapted; the adaptation will consist in removing the corresponding assignment for age_{ts}. After the removal of the source element $functionname_{s2}$, the mapping can be adapted only if one substitution is found to replace $functionname_{s1}$. In the example, we can see that there is no more information in the sources that concerns the function names (i.e. no more semantic correspondence involving $functionname_{s1}$). No substitution can be therefore found and the mapping can not be adapted for this change.

The notion of ***adaptable mappings*** is introduced to represent affected mappings that can be adapted:

Definition 4-4. Adaptable mappings. Consider a mapping defined for a target schema from a set of source schemas. If the mapping is affected by a change occurring in one of these schemas, the affected mapping is ***adaptable*** if there is at least one solution to adapt it to the new schemas. Otherwise, the mapping is not adaptable.

The mapping resulting of the adaptation is called ***adapted mapping***. Consider the change in the example of Figure 4-4 that removes the target element age_{ts}; one solution to adapt the mapping is to remove the assignment for age_{ts}. This solution gives the adapted mapping shown in Figure 4-4 (b). If there are several adaptation solutions for an affected mapping, an adapted mapping is generated for each of these solutions.

5. Mapping Adaptation for Source Evolution

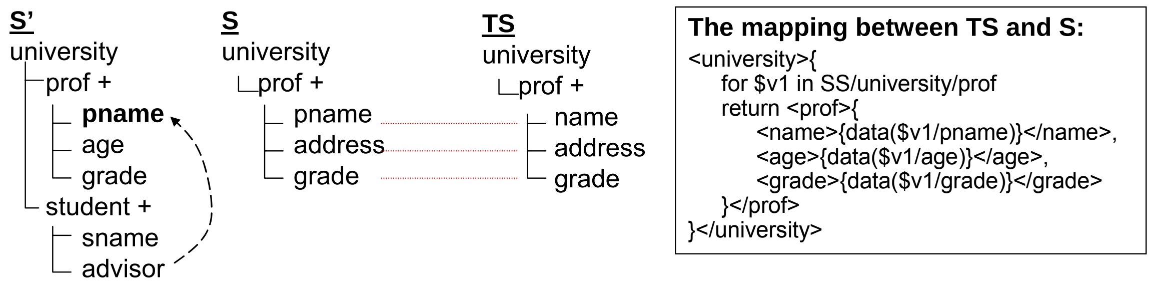
Given a mapping defined for a target schema over a set of data sources, and considering that these sources are autonomous and can freely change their content or their structure, the mapping may become obsolete as the data sources evolve. The changes occurring in the sources need to be propagated in the mappings to keep the system consistent. In this section, we propose mapping adaptation algorithms for adapting existing mappings after source changes. We consider the following changes: addition or removal of source elements, removal of source constraints, moving source subrees and renaming source elements. For each of these changes, a specific adaptation algorithm is proposed in the following subsections.

5.1. Addition of Source Elements and Source Constraints

Adding a new element in a schema tree consists in adding a leaf element without reference or key definition. The added element is associated two attributes `minOccurs` and `maxOccurs` to state the minimum number of occurrences and the maximum number of occurrences of this element with respect to its parent. If the added source element is equivalent to an element in the target schema the semantic correspondence is added. Constraints can also be added in the source schemas. A key can be added in a source schema to define a mandatory source element to be a key. We may also add a reference on a source element and it references an existing key.

The mapping is not affected with respect to these added source objects (elements, keys and constraints) because the sources components that are used in the mappings are not removed or moved. The added objects may be used for future mapping adaptations.

Consider the example in Figure 4-5. There is one target schema *TS*, one source schema *S*, and a mapping between the two schemas. Consider that the source schema S evolves and is changed to a new schema *S'*; the changes that have occurred are the addition of: the three source elements $student_{ss'}$, $sname_{ss'}$, and $advisor_{ss'}$, the addition of a key on the element $pname_{ss'}$, and the addition of a reference on the element $advisor_{ss'}$ to $pname_{ss'}$. The mapping between *TS* and *S* is not affected by these changes because none of the elements it involves have been changed.



The mapping between TS and S:

```
<university>{
    for $v1 in SS/university/prof
    return <prof>{
        <name>{data($v1/pname)}</name>,
        <age>{data($v1/age)}</age>,
        <grade>{data($v1/grade)}</grade>
    }</prof>
}</university>
```

Figure 4-5. An example of the addition of source elements and source constraints

5.2. Removal of Source Elements

We consider that removing source elements concerns the leaf elements in the schema tree such that there is no constraint defined using this element. The removal of an element *n* is followed by the removal of all the semantic correspondences involving *n*.

The removal of the element *n* invalidates all the components of the mapping that involve *n*. These components are:

- the target assignment that defines a target element from *n*;
- the join conditions involving *n*;
- the duplicate-elimination `for` clause such that *n* is in the unique union to which the clause binds its variable;
- the duplicate-elimination statement involving *n*;
- the element-binding `for` clause that binds its variable to *n*, and consequently all the other mapping components that involve a source element related to this `for` clause;
- the grouping condition involving *n*.

A mapping that has at least one of these components invalidated by the change is an affected mapping. The adaptation of this mapping will consist in repairing sequentially all the invalid mapping components. A specific process is used to repair every component. If a target assignment uses the removed element *n*, the mapping adaptation consists in substituting the invalid target assignment. Adapting a mapping for an invalid join condition requires finding a substitution to replace the invalid condition. If a duplicate-elimination `for` clause is invalidated by the removal, repairing this clause consists in redefining the element to which the variable is bound. If a duplicate-elimination or a grouping condition are invalidated by the change, a substitution has to be found to replace it. Repairing an element-binding `for` clause invalidated by the change consists in either redefining it or removing it if there is no more elements that is related to it in the mapping.

Figure 4-6 shows the procedure of mapping adaptation for a source element removal. It repairs every of the invalid components. If the mapping is not adaptable for an intermediate step, the adaptation is interrupted and the mapping is not adaptable. In this process, the mapping is always adaptable for the invalid duplicate-elimination `for` clause and the invalid duplicate-elimination statement.

In the following, we describe the repairing of the different mapping components if they are invalidated after a source element removal. For every invalid component, we characterize the adaptation solutions, and we show how to adapt the mapping with respect to the adaptation solutions.

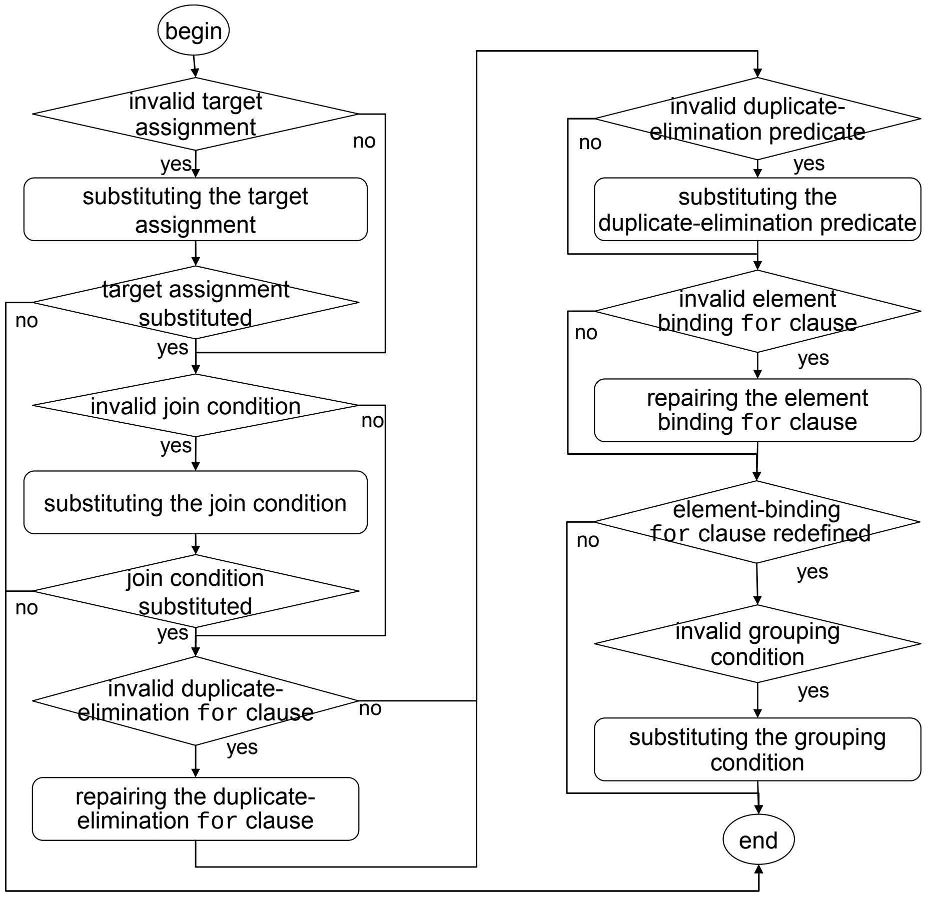

begin
invalid target assignment
yes
no
substituting the target assignment
target assignment substituted
yes
no
invalid join condition
yes
no
substituting the join condition
join condition substituted
yes
no
invalid duplicate-elimination for clause
yes
no
repairing the duplicate-elimination for clause
invalid duplicate-elimination predicate
yes
no
substituting the duplicate-elimination predicate
invalid element binding for clause
yes
no
repairing the element binding for clause
element-binding for clause redefined
yes
no
invalid grouping condition
yes
substituting the grouping condition
end

Figure 4-6. General procedure of mapping adaptation for a source element removal

5.2.1. Repairing an Invalid Target Assignment

Consider a mapping that contains an assignment from *n* to a target element *n'*. The target assignment becomes invalid if *n* is removed from its source schema. Adapting the mapping consists in finding another element to replace *n* for assigning the target element *n'*.

For repairing the target assignment, we distinguish between two cases: the target element *n'* is an optional element or it is a mandatory element. If *n'* is an optional element, it is always possible to repair the invalid target assignment by removing the target assignment for *n'*.

If *n'* is a mandatory element, we need to find another source element to assign *n'*. This source element *n''* has to be involved in a correspondence with *n'* and be related with the other source elements used to assign the target elements in the same FWR expression. For this purpose, we first introduce the notion of ***candidate join***. This joins represent a possible way to relate two source elements.

Definition 4-5. Candidate joins. Given two source elements *m* and *m'*, there is a candidate join between *m* and *m'* with the join condition *c = c'*, denoted *j[c = c'](m, m')* if the following conditions are satisfied;

- either *c* and *m* represent the same element, or *c* is monovalued with respect to *m* and vice versa;
- either *c'* and *m'* represent the same element, or *c'* is monovalued with respect to *m*' and vice versa;
- if *m* and *m'* are in the same schema, either *m* is defined as a reference of *m'* or *m'* is defined as a reference of *m*;
- if *m* and *m'* belong to different schemas, $m \cong m'$ and at least one element between *m* and *m'* is defined as a key in its schema.

For the example shown in Figure 4-7, there is a candidate join $j[authorid_{s1} = id_{s2}](address_{s1}, author_{s2})$ because (i) $address_{s1}$ is monovalued with respect to $authorid_{s2}$ and $authorid_{s2}$ is monovalued with respect to $address_{s1}$; (ii) $author_{s2}$ is monovalued with respect to id_{s2} and id_{s2} is monovalued with respect to $author_{s2}$; (iii) $authorid_{s1}$ and id_{s2} belong to different schemas: $authorid_{s1} \cong id_{s2}$ and id_{s2} is defined as a key. There is another candidate join $j[authorid_{s1} = id_{s2}](authoraddress_{s1}, name_{s2})$. There is no candidate join with the condition $name_{s1} = id_{s2}$ because these two elements are in different schemas and no correspondence relates them.

Consider a `FWR` expression *ep* that contains an invalid target assignment for *n'*. Finding a substitution to the invalid target assignment consists in finding a source element *n"* such that $n' \cong n''$ and there is an element-binding `for` clause *f* in *ep* that satisfies one of the following conditions:

- *f* binds its variable to *n"*;
- *f* binds its variable to an element *e* such that *n"* is monovalued with respect to *e* and vice versa;
- *f* binds its variable to an element *e* and there are two elements *c* and *c'* such that there is a candidate join *j[c = c'](e, n")*.

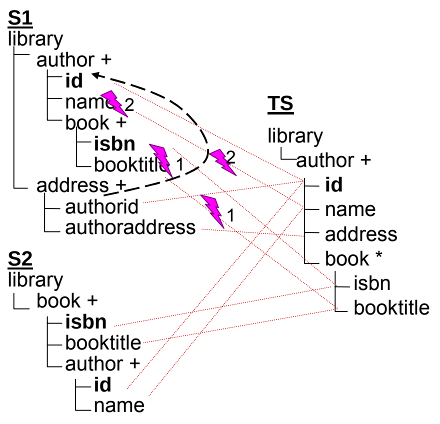
The mapping is adaptable for an invalid target assignment if at least one source element *n"* satisfies one of the above conditions. If *n"* satisfies the first or the second condition, the invalid assignment is replaced by the assignment from *n"* to *n'*; *n"* is related to the variable defined in *f*. If *n"* satisfies the third condition, the mapping adaptation consists in:

- adding an element-binding `for` clause in the `FWR` expression *ep* that binds a variable *v'* to *n"*;
- in the `where` clause of *ep*, adding a join condition *c = c'* such that *c* is related to the variable defined in *f* and *c'* is related to *v'*;
- in the `return` clause of *ep*, replacing the assignment from *n* to *n'* by the assignment from *n"* to *n*; *n"* being related to *v'*.

Figure 4-7 shows an example of repairing an invalid target assignment. There are one target schema and two source schemas. A mapping between the schemas is shown in Figure 4-7 (a). We consider that two changes occurred in the sources: the first removes the source element $booktitle_{s1}$ from *S1* and the second removes $name_{s1}$ from *S1*.

Consider the removal of the source element $booktitle_{s1}$. This change affects the original mapping because there is a `return` clause that contains an assignment from $booktitle_{s1}$ to $booktitle_{ts}$. The source element $booktitle_{s2}$ can be used for a substitution of the invalid target assignment: there is a correspondence $booktitle_{s2} \cong booktitle_{ts}$ and $booktitle_{s2}$ is monovalued with respect to $book_{s2}$ to which an element-binding `for` clause of the same `FWR` expression binds a variable. The mapping is therefore adaptable for this change and the adapted mapping is shown in Figure 4-7 (b). In the adapted mapping, the added parts are in bold.

Consider the removal of the source element *name$_{s1}$*. The mapping of Figure 4-7 (b) is affected by the change because it contains a target assignment from *name$_{s1}$* to *name$_{ts}$*. For adapting the mapping, the source element *name$_{s2}$* can be used to substitute the invalid target assignment: there is a correspondence *name$_{s2}$* ≅ *name$_{ts}$*; there is an element-binding `for` clause in the same `FWR` expression that binds a variable to the element *address$_{s1}$*; and there is a candidate join j[*id$_{s2}$* = *authorid$_{s1}$*](*name$_{s2}$*, *address$_{s1}$*). The mapping is therefore adaptable in this case and the adapted mapping is shown in Figure 4-7 (c).



(a) Original mapping:

```
<library>{
    for $ka in distinct-values(S1/library/author/id, S1/library/address/authorid)
    for $v1 in S1/library/author[id=$ka][1],
    for $v2 in S1/library/address,
    where $v1/id=$v2/authorid
    return <author>{
        <id>{data($v1/id)}</id>,
        <name>{data($v1/name)}</name>,
        <address>{data($v2/authoraddress)}</address>
        for $v4 in S1/library/author/book
        for $v5 in S2/library/book
        where $v4/ = $v1/book and $v4/isbn=$v5/isbn
        return <book>{
            <isbn>{data($v4/isbn)}</isbn>,
            <bookTitle>{data($v4/bookTitle)}</bookTitle>
        }</book>
    }</author>
}</library>
```

(b) Adapted mapping after the removal of booktitle$_{s1}$≅booktitle$_{ts}$:

```
<library>{
    for $ka in distinct-values(S1/library/author/id, S1/library/address/authorid)
    for $v1 in S1/library/author[id=$ka][1],
    for $v2 in S1/library/address,
    where $v1/id=$v2/authorid
    return <author>{
        <id>{data($v1/id)}</id>,
        <name>{data($v1/name)}</name>,
        <address>{data($v2/authoraddress)}</address>
        for $v4 in S1/library/author/book
        for $v5 in S2/library/book
        where $v4/ = $v1/book and $v4/isbn=$v5/isbn
        return <book>{
            <isbn>{data($v4/isbn)}</isbn>,
            <bookTitle>{data ($v5/bookTitle)}</bookTitle>
        }</book>
    }</author>
}</library>
```

(c) Adapted mapping after the removal of name$_{s1}$≅name$_{ts}$:

```
<library>{
    for $ka in distinct-values(S1/library/author/id, S1/library/address/authorid)
    for $v1 in S1/library/author[id=$ka][1],
    for $v2 in S1/library/address,
    for $v3 in S2/library/book/author/id
    where $v1/id=$v2/authorid and $v2/authorid=$v3
    return <author>{
        <id>{data($v1/id)}</id>,
        <name>{data($v3/PARENT::author/id)}</name>,
        <address>{data($v2/authoraddress)}</address>
        for $v4 in S1/library/author/book
        for $v5 in S2/library/book
        where $v4/ = $v1/book and $v4/isbn=$v5/isbn
        return <book>{
            <isbn>{data($v4/isbn)}</isbn>,
            <bookTitle>{data($v5/bookTitle)}</bookTitle>
        }</book>
    }</author>
}</library>
```

Figure 4-7. Mapping adaptation after the removal of two source element

As we have seen above, there are three different options for finding a substitution for the invalid target assignment. One of them states that the source element *n'* can be replaced by a source element *n"* if *n"* is related to an element-binding `for` clause through a candidate join. We can generalize this notion and consider a sequence of joins for relating *n"* to an element-binding `for` clause. For relating two elements *m* and *m'*, we can consider a sequence of joins: *j[c = c$_1$](m, e$_1$)*, *j[c$_1$ = c$_2$](e$_1$, e$_2$)*, .. , *j[c$_{k-1}$ = c$_k$](e$_{k-1}$, e$_k$)*, *j[c$_k$ = c'](e$_k$, m')*, which allows relating *m* to *e$_1$*, *e$_1$* to *e$_2$*, .. , *e$_{k-1}$* to *e$_k$*, and *e$_k$* to *m'*. We introduce the notion of ***candidate join path*** to represent this sequence of candidate joins.

Definition 4-6. Candidate join paths. Given two elements *m* and *m'*, a candidate join path between *m* and *m'* is a sequence of candidate joins: $j[c = c_1](m, e_1)$, $j[c_1 = c_2](e_1, e_2)$, .. , $j[c_{k-1} = c_k](e_{k-1}, e_k)$, $j[c_k = c'](e_k, m')$.

Using this definition, a source element *n"* can be used to assign a target element if the following conditions are satisfied:

– $n' \cong n''$;
– considering that the invalid target assignment is in the FWR expression *ep*, there is an element-binding `for` clause *f* in *ep* that binds its variable to an element *e* and there exists a candidate join path between *n"* and *e*:
  $j[c = c_1](e, e_1)$, $j[c_1 = c_2](e_1, e_2)$, .. , $j[c_{k-1} = c_k](e_{k-1}, e_k)$, $j[c_k = c'](e_k, n'')$.

Algorithm 4-1 describes the process of finding substitutions to repair an invalid target assignment. It takes as input the target element *n* concerned by the invalid target assignment and the FWR expression *ep* containing the target assignment. It produces the set of substitutions (*S*) to repair the invalid target assignment. The algorithm checks for each source element that is involved in a correspondence with *n*, if it can be related to a `for` clause in *ep*.

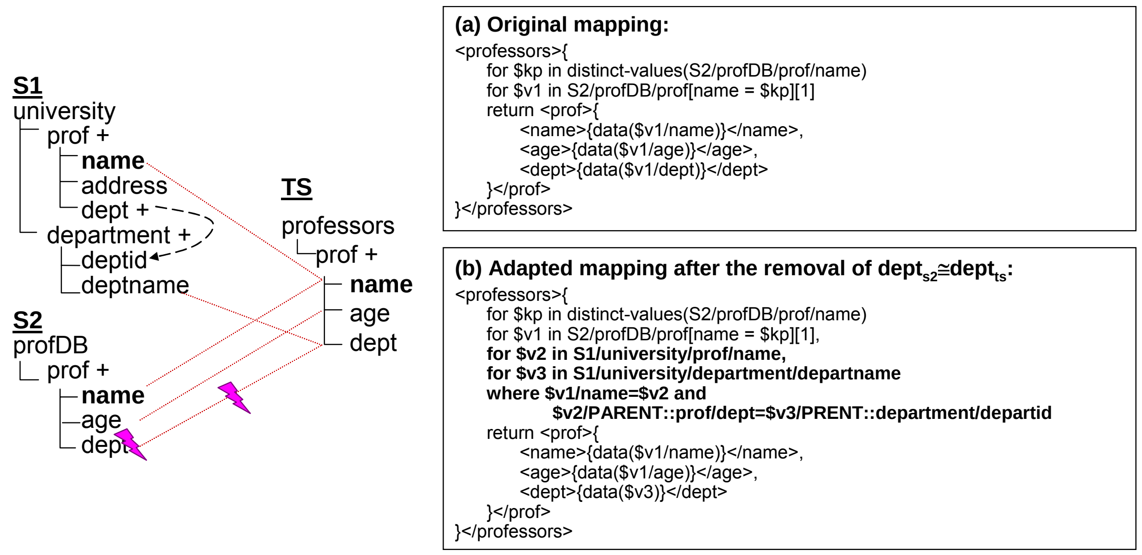
Algorithm 4-1. Finding substitutions for an invalid target assignment

```
Substitution_Search_for_an_Invalid_Target_Assignment(n, ep, S)
Begin
    S := ∅;
    for each element n' such that n ≅ n':
        for each element-binding for clause f in ep:
            if n' ≡ binding-element(f) //returns the element to which f binds its variable
                s := create-solution(n', f); //create a solution to substitute the target assignment
                S := S ∪ {s};
                break;
            e := binding-element(f);
            if monovalued(n', e) //returns true if n' is monovalued with respect to e
                and  monovalued(e, n')
                s := create-solution(n', f); //create a solution to substitute the target assignment
                S := S ∪ {s};
                break;
            else
                j := joinpath(n', e); //returns a candidate join path to relate n' and e
                if j ≠ ∅
                    s := create-solution(n', f);//create a solution to substitute the target assignment
                    S := S ∪ {s};
                    break;
End
```

If a source element *n"* can be used to repair the invalid target assignment, the mapping is adapted using the following procedure:

- adding an element-binding `for` clause in *ep* that binds a variable *v* to *n"*;
- adding an element-binding `for` clause in *ep* that binds a variable v_i to e_i ($1 \le i \le k$);
- adding in the `where` clause of *ep* a join condition $c = c_1$ with *c* being related to the variable of *f* and c_1 being related to v_1;
- adding in the `where` clause of *ep* a join condition $c_i = c_{i+1}$ ($1 \le i \le k-1$) with c_i being related to v_i and c_{i+1} being related to v_{i+1};
- adding in the `where` clause of *ep* a join condition $c_k = c'$ with c_k being related to v_k and *c'* being related to *v*;
- in the `return` clause of *ep*, replacing the invalid target assignment by an assignment from *n"* to *n*; *n"* is related to *v*.

Figure 4-8 shows an example of repairing an invalid target assignment. A mapping between a target schema and two source schemas is shown in Figure 4-8 (a). Consider that the source element $dept_{s2}$ is removed. The mapping is affected because it contains an assignment from $dept_{s2}$ to $dept_{ts}$. The source element $deptname_{s1}$ can be used to repair the invalid target assignment: there is a correspondence $deptname_{s1} \cong dept_{ts}$; and there is a candidate join path j[$name_{s2} = name_{s1}$]($prof_{s2}$, $name_{s1}$), j[$dept_{s1} = deptid_{s1}$]($name_{s1}$, $deptname_{s1}$). The mapping is adapted using this candidate join path and the result is shown in Figure 4-8 (b).



(a) Original mapping:

```
<professors>{
    for $kp in distinct-values(S2/profDB/prof/name)
    for $v1 in S2/profDB/prof[name = $kp][1]
    return <prof>{
        <name>{data($v1/name)}</name>,
        <age>{data($v1/age)}</age>,
        <dept>{data($v1/dept)}</dept>
    }</prof>
}</professors>
```

(b) Adapted mapping after the removal of $dept_{s2} \cong dept_{ts}$:

```
<professors>{
    for $kp in distinct-values(S2/profDB/prof/name)
    for $v1 in S2/profDB/prof[name = $kp][1],
    for $v2 in S1/university/prof/name,
    for $v3 in S1/university/department/departname
    where $v1/name=$v2 and
          $v2/PARENT::prof/dept=$v3/PRENT::department/departid
    return <prof>{
        <name>{data($v1/name)}</name>,
        <age>{data($v1/age)}</age>,
        <dept>{data($v3)}</dept>
    }</prof>
}</professors>
```

Figure 4-8. Mapping adaptation after the removal of a source element

5.2.2. Replacing the Invalid Join Conditions

If a source element *n* is removed from a source schema, the join condition involving *n* becomes invalid. Removing a join condition from the `where` clause of a `FWR` expression consists in removing a join that relates two element-binding `for` clauses *f* and *f'*. Adapting the mapping to substitute this removed join condition consists in finding a way to relate *f* and *f'*.

Consider that *f* binds its variable to *m* and *f'* binds its variable to *m'*. Finding a way to relate *f* and *f'* consists in finding a candidate join path between the two elements *m* and *m'*: $j[c = c_1](m, e_1)$, $j[c_1 = c_2](e_1, e_2)$, .. , $j[c_{k-1} = c_k](e_{k-1}, e_k)$, $j[c_k = c'](e_k, m')$. If there is at least one candidate join path, the mapping is adaptable.

Algorithm 4-2 describes the process of finding substitutions to repair an invalid join condition. It takes as input the two element-binding `for` clauses *f* and *f'* that were related by the invalid join condition and the FWR expression *ep* containing the condition. It produces the set of substitutions (*S*) to repair the invalid target assignment. The algorithm search for all the candidate join paths between the variable defined by *f* and the variable defined by *f'*.

Algorithm 4-2. Finding substitutions for an invalid join condition

```
Substitution_Search_for_an_Invalid_Join_Condition(f, f', ep, S)
Begin
    S := ∅;
    e := binding-element(p); //returns the element to which p binds its variable
    e := binding-element(q);
    for each j  in joinpaths(e, e') //returns the candidate join paths between n' and e
        s := create-solution(j);//create a solution to substitute the invalid join condition
        S := S ∪ {s};
End
```

Consider the `FWR` expression *ep* that contains the invalid join condition. For every candidate join path $j[c = c_1](m, e_1)$, $j[c_1 = c_2](e_1, e_2)$, .. , $j[c_{k-1} = c_k](e_{k-1}, e_k)$, $j[c_k = c'](e_k, m')$ that satisfies the above conditions by relating the two for clauses *f* and *f'*, we substitute the invalid join condition by:

- adding an element-binding `for` clause in *ep* such that it binds a variable v_i to e_i $(1 \leq i \leq k)$;
- adding in the `where` clause of *ep* a join condition $c_i = c_{i+1}$ $(1 \leq i \leq k-1)$; c_i is related to v_i and c_{i+1} is related to v_{i+1};
- adding in the `where` clause of *ep* a join condition $c = c_1$; c is related to the variable defined by *f* (related to *m*) and c_1 is related to v_1;
- adding in the `where` clause of *ep* a join condition $c_k = c'$; c_k is related to v_k and c' is related to the variable defined *f'* (related to *m'*);

Figure 4-9 shows an example of mapping adaptation after the removal of a source element. There are one target schema and three source schemas. A mapping shown Figure 4-9 (a) derives instances of the target schema from the source schemas S1 and S3. Consider that the source element id_{s3} is removed with the correspondence $id_{s3} \cong id_{ts}$. This change affects the mapping by invalidating three components: the join condition *$v1/id = $v2/id*, the duplicate-elimination `for` clause that defines the variable *$au*, and the element-binding `for` clause that contains a duplicate-elimination statement *[id=$au][1]*. The affected components are in bold and italic in Figure 4-9 (a). We adapt the mapping sequentially for the three invalidated components. We here illustrate the repairing of the join condition *$v1/id = $v2/id*. The repairing of the other components is shown in the following sections.

The invalid join condition *$v1/id=$v2/id* relates the two element-binding `for` clauses of the `FWR` expression. Consider the candidate join path j[$name_{s1}$=$name_{s2}$]($author_{s1}$, $name_{s2}$),j[$name_{s2}$=$name_{s3}$]($name_{s2}$, $author_{s3}$). It relates the two elements $author_{s1}$ and $author_{s3}$ to which these two `for` clauses bind their variable respectively. Therefore it can be used to substitute the invalidate join condition. Figure 4-9 (b) shows the repaired mapping using the candidate join path; the added components are in bold.

5.2.3. Repairing the Duplicate-Elimination `for` Clause

Consider a duplicate-elimination `for` clause; it is invalidated by the removal of a source element *n* if it uses *n* to bind the variable of the `for` clause.

The invalid duplicate-elimination `for` clause has to be redefined in this case. As we assumed that every invalid duplicate-elimination `for` clause is defined for a key defined on a target element *n*, we search for all the source elements that can be used to assign *n'* in this `FWR` expression; a source element *n* has to satisfy the following conditions for being an adaptation solution:

- $n \cong n'$;
- there is an element-binding `for` clause *f* in the same `FWR` expression such that either *f* binds its variable to *n* or *n* is monovalued with respect to the element to which *f* binds its variable.

The mapping is adaptable if at least one element satisfies the condition. We can prove that the mapping is always adaptable in this case. Every duplicate-elimination `for` clause is defined for a key defined on a mandatory target element. Consider the process of mapping adaptation for a source element removal. Before repairing the element-binding `for` clause, we first repair the invalid target assignment if the removed source element is used for a target assignment. The process only continues if there is a way to define the target element. Therefore at the moment of re-defining the element-binding `for` clause, there is at least one source element that satisfies the condition to define `for` clause, which is the one used to assign the target element in the `return` clause.

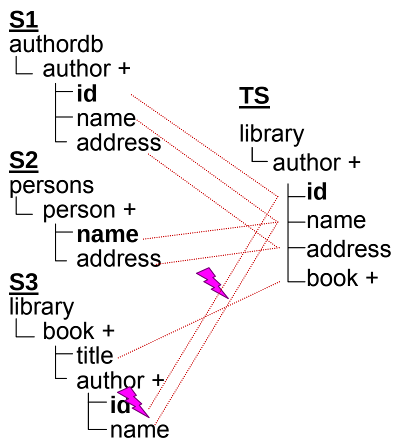
Algorithm 4-3. Finding elements for an invalid duplicate-elimination `for` clause

```
Element_Search_for_an_Duplicate_Elimination_for_clause(n, ep, E)
Begin
    E := ∅;
    for each element n' such that n ≅ n':
        for each element-binding for clause f in ep:
            if n' ≡ binding-element(f) //returns the element to which f binds its variable
                E := E ∪ {n'};
                break;
            m := binding-element(f);
            if monovalued(n', m) //returns true if n' is monovalued with respect to e
                E := E ∪ {n'};
End
```

Algorithm 4-3 describes the process of finding elements to repair an invalid duplicate-elimination `for` clause. It takes as input the target element *n* for which the invalid `for` clause has been defined and the FWR expression *ep* containing the `for` clause. It produces the set of elements (*E*) that will be used to redefine the invalid `for` clause. The algorithm checks for each source element *n'* that is involved in a correspondence with *n*, if there is a `for` clause in *ep* that binds the variable either to *n'* or to another element with respect to which *n'* is monovalued.

The set *E* of all the elements returned by Algorithm 4-3 constitutes a solution to repair the `for` clause. The repairing process consists in re-defining the duplicate-elimination `for` clause to bind its variable to the distinct union of the elements in *E*.

Consider the example shown in Figure 4-9. Three components of the original mapping have been invalidated by the removal of the source element id_{s3}. Among these invalid components, the invalid join condition has first been substituted and the result is shown in Figure 4-9 (b). Afterwards, we repair the invalid duplicate-elimination `for` clause that defines the variable *$au*. One element id_{s1} is satisfies the conditions to be an adaptation solution. It is therefore used to redefine the duplicate-elimination `for` clause and the result is shown in Figure 4-9 (c).



(a) Original mapping:

```
<library>{
    for $au in distinct-values(S1/authordb/author/id,
                               S3/library/book/author/id)
    for $v1 in S1/authordb/author,
    for $v2 in S3/library/bbok/author[id=$au][1]
    where $v1/id=$v2/id
    return <author>{
        <id>{data($v1/id)}</id>,
        <name>{data($v2/name)}</name>,
        <address>{data($v1/address)}</address>
        for $v4 in S3/library/book
        where $v4/author = $v2
        return <book>{ data($v4/title) }</book>
    }</author>
}</library>
```

(b) Adapted mapping after having found a substitution for the invalid value-join condition:

```
<library>{
    for $au in distinct-values(S1/authordb/author/id,
                               S3/library/book/author/id)
    for $v1 in S1/authordb/author,
    for $v2 in S3/library/bbok/author[id=$au][1],
    for $v3 in S2/persons/person/name
    where $v1/name=$v3 and $v2/name=$v3
    return <author>{
        <id>{data($v1/id)}</id>,
        <name>{data($v2/name)}</name>,
        <address>{data($v1/address)}</address>
        for $v4 in S3/library/book
        where $v4/author = $v2
        return <book>{ data($v4/title) }</book>
    }</author>
}</library>
```

(c) Adapted mapping after having defined the invalid grouping for clause:

```
<library>{
    for $au in distinct-values(S1/authordb/author/id)
    for $v1 in S1/authordb/author,
    for $v2 in S3/library/bbok/author[id=$au][1],
    for $v3 in S2/persons/person/name
    where $v1/name=$v3 and $v2/name=$v3
    return <author>{
        <id>{data($v1/id)}</id>,
        <name>{data($v2/name)}</name>,
        <address>{data($v1/address)}</address>
        for $v4 in S3/library/book
        where $v4/author = $v2
        return <book>{ data($v4/title) }</book>
    }</author>
}</library>
```

(d) Adapted mapping after having finding a substitution for the invalid grouping condition:

```
<library>{
    for $au in distinct-values(S1/authordb/author/id)
    for $v1 in S1/authordb/author[id=$au][1],
    for $v2 in S3/library/bbok/author,
    for $v3 in S2/persons/person/name
    where $v1/name=$v3 and $v2/name=$v3
    return <author>{
        <id>{data($v1/id)}</id>,
        <name>{data($v2/name)}</name>,
        <address>{data($v1/address)}</address>
        for $v4 in S3/library/book
        where $v4/author = $v2
        return <book>{ data($v4/title) }</book>
    }</author>
}</library>
```

Figure 4-9. Another example of mapping adaptation for a source element removal

5.2.4. Substituting the Invalid Duplication-Elimination Statement

The removal of the source element *n* invalidates the duplicate-elimination statement involving *n*. We recall that a duplicate-elimination variable has been defined for a target element defined as a key, and duplicate-elimination statements are used to ensure that the returned instances of the mapping will not contain duplicates with respect to this target element. Adapting the mapping therefore consists in finding a substitution to replace the invalid statement.

Consider the `FWR` expression *ep* of the invalid duplicate-elimination statement, and the target element *n'* for which the duplicate-elimination statement is defined. Finding a substitution for the statement consists in finding an element *n"* and another element-binding `for` clause *f* in *ep* such that:

- $n' \cong n''$;
- *f* either binds its variable to *n"* or binds its variable to an element *e* such that *e* is monovalued with respect to *n"* and vice versa.

If these conditions are satisfies, the element *n"* and the `for` clause *f* together are an adaptation solution. If there is at least one element satisfying the conditions, then the mapping is adaptable. Notice that for the same reason as we mentioned for repairing duplicate-elimination `for` clauses, there is always at least one element satisfying the condition and the mapping is always adaptable.

The way of finding an element to substitute a duplicate-elimination statement is the same as the one to redefine a duplicate-elimination for clause presented in Algorithm 4-3.

For every combination of the element *n"* and the `for` clause *f* that is an adaptation solution, repairing the mapping consists in:

- adding in *f* a duplicate-elimination statement that involves *n"* and the same duplicate-elimination variable used in the invalid one;
- removing the invalid duplicate-elimination statement.

Consider again the example of Figure 4-9. The last invalid mapping component that needs to be repaired is the duplicate-elimination statement *[id=$au][1]*. The element id_{s1} and the `for` clause defining the variable *$v1* together are an adaptation solution. The statement *[id=$au][1]* is added in this `for` clause to replace the invalid one; the adapted mapping is shown in Figure 4-9 (d). All the added components are in bold.

5.2.5. Repairing the Invalid Element-Binding `for` Clause

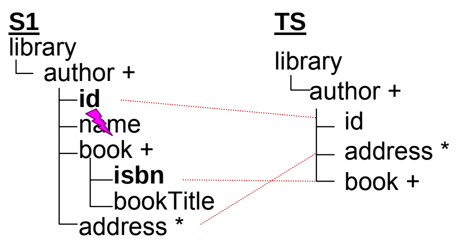
If a source element *n* is removed, the element-binding `for` clause *f* that binds its variable to *n* is invalidated. Consequently, all the other mapping components that involve an element related to the variable of *f* are invalidated; these mapping components may be join conditions, grouping conditions and target assignments.

Consider the set *N* of elements related to the variable of the invalid `for` clause *f* and that are used either in a target assignment or in a join condition. We distinguish between the two cases depending on if *N* is empty or not.

If *N* is empty, the invalid `for` clause is the only element-binding `for` clause of the `FWR` expression and there is no target assignment. This case is possible if the `FWR` expression populates a single optional element. After the removal of the source element, we only remove its assignment because the target element is optional. In this case, the adaptation consists in removing the whole `FWR` expression containing the invalid `for` clause.

If *N* is not empty, the invalid `for` clause *f* was used either to relate other `for` clauses or to assign a target element; it therefore needs to be re-defined to bind its variable to another element. Consider an element *e* in *N* such that the other elements of *N* are monovalued with respect to *e*; the invalid `for` clause is re-defined to binds the variable to *e*.

Once the invalid `for` clause is re-defined, all the other elements of *N* are changed to be related to *n'*, except for the grouping conditions that involve *n*.



(a) Original mapping:

```
<library>{
    for $sp1 in S1/library/author/name
    return <author>{
        <id>{data($sp1/PARENT::author/id)}</id>
        for $sp2 in S1/library/author/book/isbn
        where $sp1/PARENT::author/book/isbn = $sp2
        return <book>{ data($sp2) }</book>
        for $sp3 in S1/library/author/address
        where $sp1 = $sp3/PARENT::author/name
        return <address>{data($sp3)}</address>
    }</author>
}</library>
```

(b) Repaired mapping for the invalid for clause sp1:

```
<library>{
    for $sp1 in S1/library/author/id
    return <author>{
        <id>{data($sp1)}</id>
        for $sp2 in S1/library/author/book/isbn
        where $sp1/PARENT::author/book/isbn = $sp2
        return <book>{ data($sp2) }</book>
        for $sp3 in S1/library/author/address
        where $sp1 = $sp3/PARENT::author/name
        return <address>{data($sp3)}</address>
    }</author>
}</library>
```

(c) Repaired mapping for the invalid join condition:

```
<library>{
    for $sp1 in S1/library/author/id
    return <author>{
        <id>{data($sp1)}</id>
        for $sp2 in S1/library/author/book/isbn
        where $sp1/PARENT::author/book/isbn = $sp2
        return <book>{ data($sp2) }</book>
        for $sp3 in S1/library/author/address
        where $sp1 = $sp3/PARENT::author/id
        return <address>{data($sp3)}</address>
    }</author>
}</library>
```

Figure 4-10. A third example of mapping adaptation for a source element removal

Consider the example of Figure 4-10 that contains one target schema and one source schema. A mapping between them is shown in Figure 4-10 (a). The removal of the source element $name_{s1}$ affects two mapping components: the element-binding `for` clause defining the variable *$sp1* and the grouping condition *$sp1 = $sp3/PARENT::author/name*. Two other mapping components are indirectly affected because they involve an element related to *$sp1*: the target assignment using id_{s1} and the grouping condition *$sp1/PARENT::author/book/isbn = $sp2*. All the invalid mapping components are in bold and italic. We illustrate here the adaptation of the mapping by repairing the invalid `for` clause and the two mapping components that involves an element related to *$sp1*. The repairing of the grouping condition involving $name_{s1}$ will be presented in the following section.

There is no join condition involving an element related to *$sp1*. The element id_{s1} is the only one related to *$sp1* and it is used for a target assignment. The invalid `for` clause is therefore re-defined to bind the variable to id_{s1}. The two components indirectly invalidated are also repaired: the path for the element related to *$sp1* is updated with respect to the new element to which *$sp1* is bound. Figure 4-10 (b) gives the result of the adaptation; the repaired components are in bold in the new mapping.

5.2.6. Substituting the Invalid Grouping Conditions

The grouping conditions involving *n* are invalidated by the removal of the element *n*. The mapping needs to be adapted; the adaptation consists in finding a substitution for the grouping condition that groups the instances of *ep* by instances *ep'*. This consists in finding two elements *e* and *e'* such that:

– there is an element-binding `for` clause *f* in *ep* that binds its variable to *e*;
– there is an element-binding `for` clause *f'* in *ep'* that binds its variable to *e'*;
– either *e* is monovalued with respect to *e'*, or *e'* is monovalued with respect to *e*.

If there is a pair of `for` clauses satisfying the conditions, the mapping is adaptable. For every pair of elements *e* and *e'*, the mapping is adapted by adding in the `where` clause of *ep* a new grouping condition that involves either *e'* or *e'* related to *f* and *f'*.

Consider again the example in Figure 4-10. The mapping in Figure 4-10 (b) still contains an invalid grouping condition *$sp1 = $sp3/PARENT::author/name*. To repair it, we consider a `for` clause that binds a variable to id_{s1} and a `for` clause that binds a variable to $address_{s1}$ and id_{s1} is monovalued with respect to *address$_{s1}$*; there is therefore a solution to substitute the invalid grouping condition. The mapping is adapted by adding the grouping condition *$sp1 = $sp3/PARENT::author/id* to replace the invalid one. Figure 4-10 (c) shows the adapted mapping.

Algorithm 4-4 described the process of finding substitutions to repair an invalid grouping condition. It takes as input the FWR expression *ep* containing the invalid grouping condition and the FWR expression in which *ep* is nested. It produces the set of substitutions (*S*) that will be used to substitute the grouping condition. For every element-binding `for` clause *f* in *ep* and every element-binding `for` clause *f'* in *ep'* such that *f* and *f'* bind their variables respectively to the elements *e* and *e'*, the algorithm checks if *e* is monovalued with respect to *e'*.

Algorithm 4-4. Finding substitutions for an invalid grouping condition

```
Substitution_Finding_for_an_Invalid_Grouping_Solution(ep, ep', S)
Begin
    S := ∅;
    for each element-binding for clause f in ep:
        for each element-binding for clause f' in ep':
            m := binding-element(f);
            m' := binding-element(f');
            if monovalued(m, m') or monovalued(m', m)
                s := create-solution(f, f'); //create a solution to substitute the grouping condition
                S := S ∪ {s};
End
```

5.3. Removal of Source Constraints

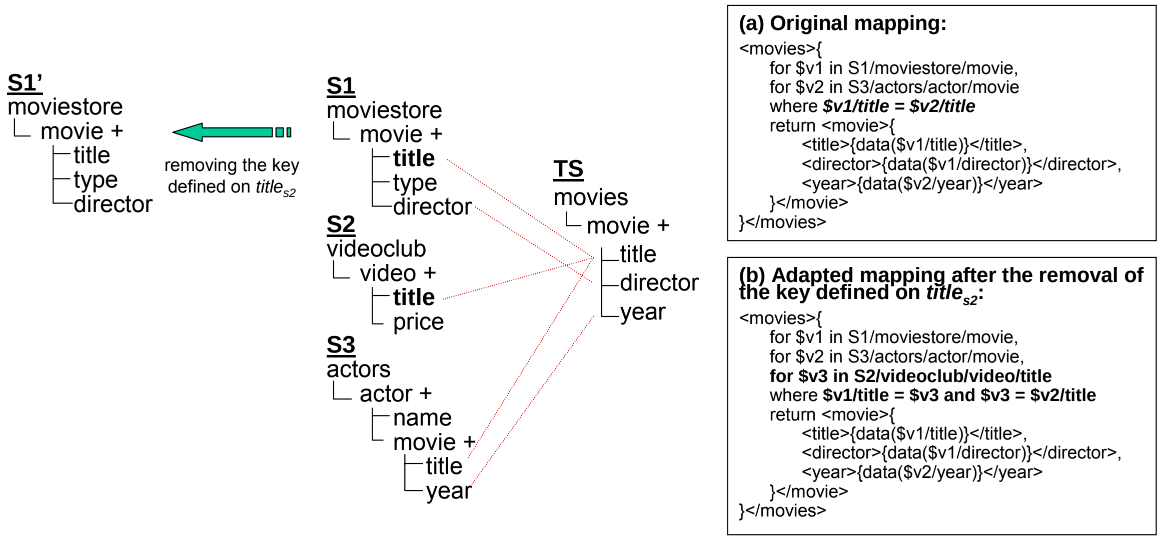
We consider that the evolutions occurring in the sources may also concern the integrity constraints, such as keys and referential constraints. Depending on whether the generation of the original mapping has been performed using these constraints; their removal may affect the mapping or not. If the mapping has been generated by an automatic approach based on the source constraints, removing them will affects the mapping. If the mapping has been generated manually or generated by an automatic approach that does not use the source constraints, removing such constraints will not affects the mapping.

Removing a reference consists in releasing the constraints on the elements defining the reference. If the intra-source joins in the mapping were inferred based on references during the mapping generation (e.g. [KX05d, PVM03]), removing them may affect the mapping. If the intra-source joins were not inferred based on references during the generation, removing references will not affect the mapping. Consider the elements *n* and *n'* of the same source schema, a key *k* defined on *n'* and a reference *r* defined on *n* to reference *k*. Removing *r* affects the mapping if there is a join condition between *n* and *n'* in one of its `where` clauses and if this join has been inferred based on the removed reference.

Removing source keys only concerns keys without references. If the mapping generation assumes that inter-source joins are based on key definitions (e.g. [KX05d]), key removal may affect the mapping. If the generation (e.g. [PVM05]) does not make such assumption, removing keys does not affect the mapping. Consider a key defined on the element *n* being removed from its source. The change affects the mapping if it contains a join condition between *n* and another element *n'* such that *n'* is in another source and it is not defined as a key, and the join is inferred based on the keys.

For both of the two cases, if the mapping is affected, the mapping adaptation consists in substituting the invalidated join condition *n=n'*. The process of finding a substitution to replace an invalid join condition is presented Section 5.2.2; the join condition is invalid because of the removal of one of the two elements involved in the condition. In this case, a join condition is invalidated because source constraint is removed, but the process of finding a substitution is the same as the one described in Section 5.2.2.

An example is shown in Figure 4-11 to illustrate the adaptation in this case. There are a target schema, three source schemas and a mapping is shown in Figure 4-11 (a). Suppose that this mapping has been generated using an automatic tool based on the key constraints. Consider a source change that removes the key defined on *title$_{s2}$*. The mapping is affected by the change because it contains a join condition (in bold and italic in Figure 4-11 (a)) involving *title$_{s1}$* and *title$_{s3}$*; *title$_{s3}$* is not a key. Adapting the mapping consists in finding a candidate join path to relate the two `for` clauses in the `FWR` expression. Since there is a candidate join path *j[title$_{s1}$, title$_{s2}$](movie$_{s1}$, video$_{s2}$), j[title$_{s2}$, title$_{s3}$](video$_{s2}$, movie$_{s3}$)* between the two elements *movie$_{s1}$* and *movie$_{s3}$*, the mapping is adaptable and the adapted mapping is shown in Figure 4-11 (b). The added components are in bold.



(a) Original mapping:

```
<movies>{
    for $v1 in S1/moviestore/movie,
    for $v2 in S3/actors/actor/movie
    where $v1/title = $v2/title
    return <movie>{
        <title>{data($v1/title)}</title>,
        <director>{data($v1/director)}</director>,
        <year>{data($v2/year)}</year>
    }</movie>
}</movies>
```

(b) Adapted mapping after the removal of the key defined on *title$_{s2}$*:

```
<movies>{
    for $v1 in S1/moviestore/movie,
    for $v2 in S3/actors/actor/movie,
    for $v3 in S2/videoclub/video/title
    where $v1/title = $v3 and $v3 = $v2/title
    return <movie>{
        <title>{data($v1/title)}</title>,
        <director>{data($v1/director)}</director>,
        <year>{data($v2/year)}</year>
    }</movie>
}</movies>
```

Figure 4-11. Mapping Adaptation for a source constraint removal

5.4. Moving Source Subtrees

Moving a subtree from its original place to a new place in the same schema consists in:

- removing the elements of the subtree from the original place, as well as all the correspondences and constraints that involves one of its element;
- and adding the same elements of the subtree at the new place with the new correspondences and constraints for these elements.

Moving a leaf element *n* with its correspondences and its constraints can be considered to be a special case of moving a subtree that contains a single element.

For each moved element *n*, if there was a correspondence relating *n* at the original place with an element *m*, there is an correspondence between *n* at the new place and *m*. To illustrate the process of re-defining the constraints, consider the example of a subtree *t* being moved to be a child of an element *p'*. For every constraint defined on elements in *t* and having the scope element in *t*, a new constraint is defined to imply the same elements in *t* at the new place. Consider that a constraint is defined on an element of *t* and it has the scope element *c* being outside of *t*. If *c* is monovalued with respect to *p'*, a new constraint will be defined on the same element at the new place; the scope of the new constraint will be the first common ascendant of *c* and *p'*. If *c* is multivalued with respect to *n*, no constraint will be defined from it for *t* at the new place. The rules used to re-define constraints after moving a subtree are shown in Figure 4-12.

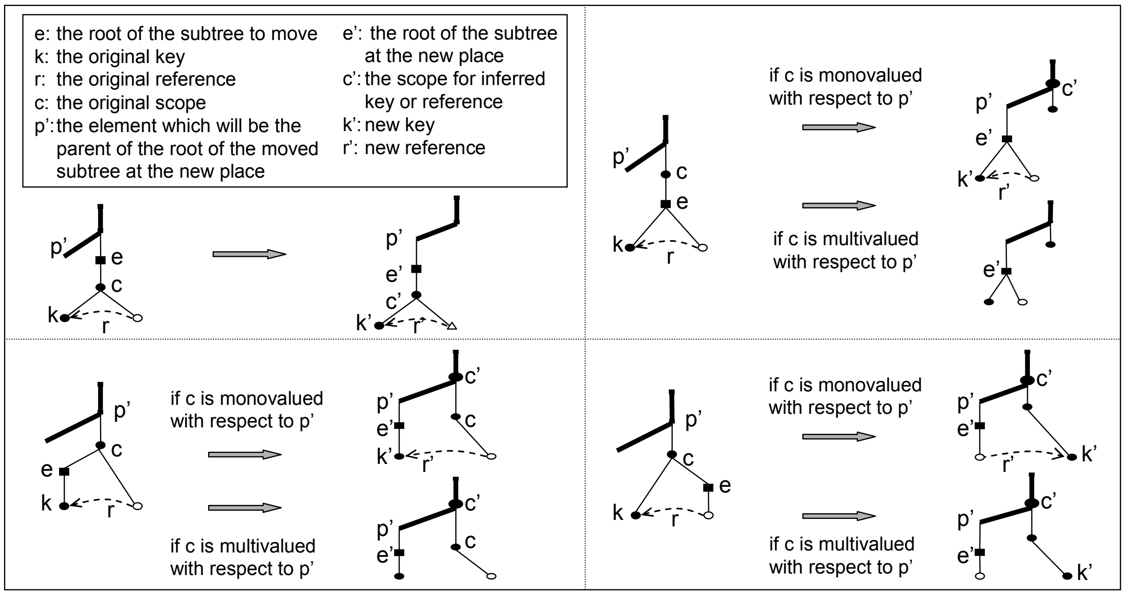

e: the root of the subtree to move
k: the original key
r: the original reference
c: the original scope
p': the element which will be the parent of the root of the moved subtree at the new place
e': the root of the subtree at the new place
c': the scope for inferred key or reference
k': new key
r': new reference
if c is monovalued with respect to p'
if c is multivalued with respect to p'
p'
e
c
k
r
e'
c'
k'
r'

Figure 4-12. Re-defining keys and references after moving a subtree

Moving a source subtree *t* can be considered as a composition of the following changes:

- removing all the source references that reference to a key defined on an element of *t*;
- removing all the source keys and source references that involve an element in *t*;
- removing all the source element of *t* in a bottom up order with their correspondences;
- adding the elements of *t* at the new place in a top to down order with the new correspondences; the cardinality of elements at the new place is the same as the ones at the original place.
- adding the new source keys for the elements of the added subtree;
- adding the new source references for the elements of the added subtree.

The mapping components that are invalidated by a source subtree moving are the components that are invalidated by one of its composing changes. Notice that all the invalid changes in this case are the changes invalidated by a removal. The source addition changes never affect a mapping.

5.5. Renaming of Source Elements

Renaming source elements concerns all leaf or intermediate elements in the source schemas and it consists in changing the tag name of the element. The change affects all the components involving this element and the adaptation of the change consists simply in changing the name of this element used in the mapping. The mapping is always adaptable in this case.

6. Mapping Adaptation for Target Evolution

Beside evolution occurring in the sources, some changes may also occur in the target schema. These changes reflect the evolution of the users' needs. In this section, we present the adaptation of the mappings in the case of changes occurring in the target schema. We consider the following changes: the addition or the removal of a target element, the addition or the removal of a key in the target schema, moving target subtrees and renaming target elements.

6.1. Removal of Target Elements

We consider that removing a target element concerns only a leaf element *n* such that there is no key defined on *n*. The removal affects the mapping if it contains an assignment for *n*.

The mapping is always adaptable for target element removal. There are two cases to consider depending on the cardinality of the removed element:

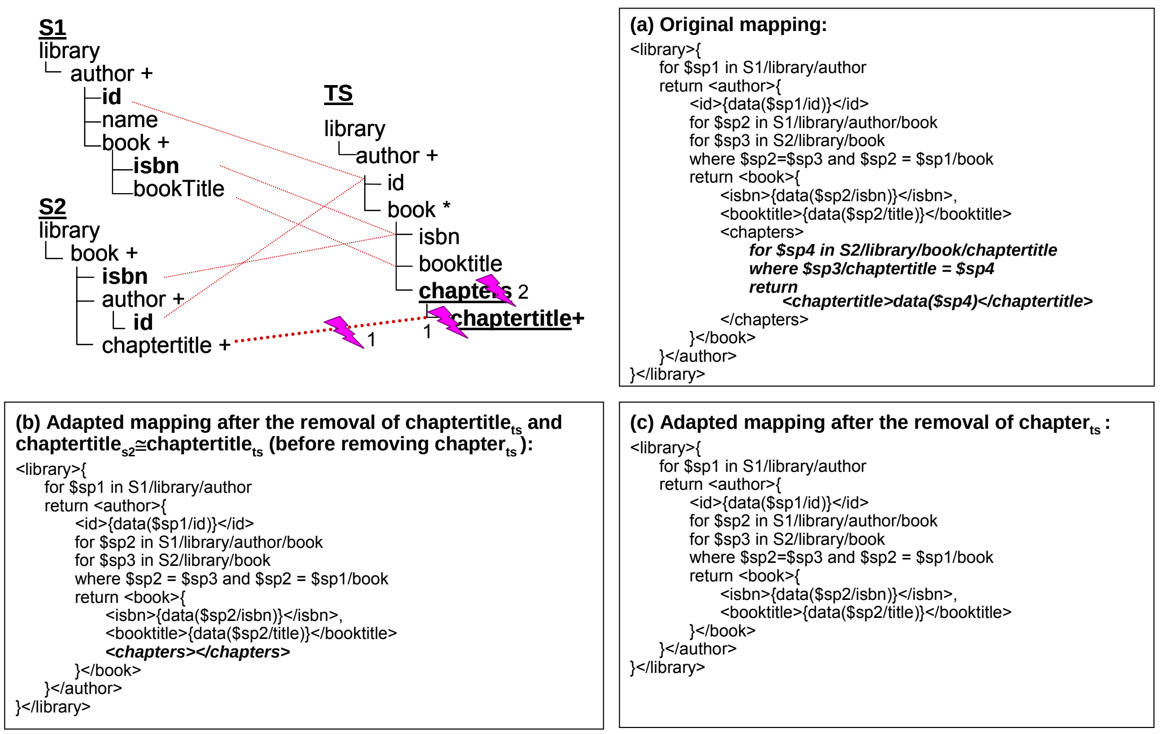
- if the removed element is monovalued, the adaptation consists in removing the corresponding assignment.
- if the removed element is multivalued, the adaptation will remove the whole `FWR` expression assigning the removed element.

Figure 4-13 shows an example of mapping adaptation after the removal of two target elements. We consider the following changes in the target schema: the element *chaptertitle*$_{ts}$ is first removed, and then the element *chapters*$_{ts}$ is removed. The original mapping of Figure 4-13 (a) is affected by the first change. Since *chaptertitle*$_{ts}$ is multivalued, the mapping is adapted by removing the whole `FWR` expression for assigning the element *chaptertitle*$_{ts}$ (in bold and italic in Figure 4-13 (a)). The result mapping of the adaptation is given in Figure 4-13 (b). This mapping is still affected by the removal of the element *chapters*$_{ts}$, which is monovalued; the assignment for *chapter*$_{ts}$ is therefore removed and the adapted mapping is shown in Figure 4-13 (c).

6.2. Addition of Target Element

Adding a new target element *n* consists in adding *n* as a leaf element in the target schema. The element *n* is associated with the two attributes `minOccurs` and `maxOccurs` to inform respectively about the minimum and the maximum number of its occurrences with respect to its parent. Recall that with these two attributes, the element is either monovalued or multivalued and it is either mandatory or optional. Some semantic correspondences are added to relate *n* with its equivalent elements in the sources. If *n* has no equivalent source element, there is no addition of semantic correspondence.

The mapping is affected if *n* is mandatory and the mapping contains an assignment for the parent of *n*. The process of repairing the mapping is also influenced by whether the added element is a text element or a non-text element. If the added element is a non-text element, we do not need to assign it from the sources. Otherwise, it needs to be assigned from a source element.



(a) Original mapping:

```
<library>{
    for $sp1 in S1/library/author
    return <author>{
        <id>{data($sp1/id)}</id>
        for $sp2 in S1/library/author/book
        for $sp3 in S2/library/book
        where $sp2=$sp3 and $sp2 = $sp1/book
        return <book>{
            <isbn>{data($sp2/isbn)}</isbn>,
            <booktitle>{data($sp2/title)}</booktitle>
            <chapters>
                for $sp4 in S2/library/book/chaptertitle
                where $sp3/chaptertitle = $sp4
                return
                    <chaptertitle>data($sp4)</chaptertitle>
            </chapters>
        }</book>
    }</author>
}</library>
```

(b) Adapted mapping after the removal of chaptertitle$_{ts}$ and chaptertitle$_{s2}$≅chaptertitle$_{ts}$ (before removing chapter$_{ts}$):

```
<library>{
    for $sp1 in S1/library/author
    return <author>{
        <id>{data($sp1/id)}</id>
        for $sp2 in S1/library/author/book
        for $sp3 in S2/library/book
        where $sp2 = $sp3 and $sp2 = $sp1/book
        return <book>{
            <isbn>{data($sp2/isbn)}</isbn>,
            <booktitle>{data($sp2/title)}</booktitle>
            <chapters></chapters>
        }</book>
    }</author>
}</library>
```

(c) Adapted mapping after the removal of chapter$_{ts}$:

```
<library>{
    for $sp1 in S1/library/author
    return <author>{
        <id>{data($sp1/id)}</id>
        for $sp2 in S1/library/author/book
        for $sp3 in S2/library/book
        where $sp2=$sp3 and $sp2 = $sp1/book
        return <book>{
            <isbn>{data($sp2/isbn)}</isbn>,
            <booktitle>{data($sp2/title)}</booktitle>
        }</book>
    }</author>
}</library>
```

Figure 4-13. Mapping Adaptation for a target element removal

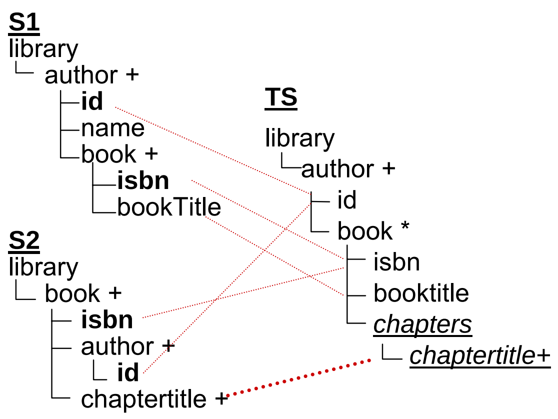
The rest of the section describes the mapping adaptations in three different cases: the element is a non-text element, the element is a monovalued text element or the element is a multivalued text element. The adaptation of the mapping in each of these situations is described in the following subsections.

6.2.1. Repairing the Mapping after the Addition of a Mandatory Non-Text Element

If the element *n* added in the target schema is a mandatory non-text element, the mapping is always adaptable and the adaptation consists in adding in the mapping the element *n* without any assignment inside; the element is nested in the assignment for the parent of *n*.

If *n* is monovalued, the adaptation consists in adding an empty assignment for *n* in the `FWR` expression of the subtree of the parent of *n*. If *n* is multivalued, the added element is represented by a new `FWR` expression for the subtree containing this single element *n*: it only contains an empty assignment for *n* and it is nested inside the `FWR` expression of the subtree of the parent of *n*.

Figure 4-14 shows an example of the addition of a mandatory non-text element. This is the reverse situation as the one of the example shown in Figure 4-13. There is one target schema and two source schemas. A mapping between these schemas is shown in Figure 4-14 (a). Consider two elements that are successively added in the target schema: *chapters$_{ts}$* and then *chaptertitle$_{ts}$*. In Figure 4-14, they are represented in bold and underlined. In this section, we consider the mapping adaptation for the addition of the element *chapters$_{ts}$*. It is mandatory and the original mapping shown in Figure 4-14 (a) is affected by the change. Since *chapters$_{ts}$* is a non-text element, an element for *chapters$_{ts}$* is nested in the assignment for the parent *book$_{ts}$*. The result of the adaptation is shown in Figure 4-14 (b) and the added component is in bold.



(a) Original mapping:

```
<library>{
    for $sp1 in S1/library/author
    return <author>{
        <id>{data($sp1/id)}</id>
        for $sp2 in S1/library/author/book
        for $sp3 in S2/library/book
        where $sp2 = $sp3 and $sp2 = $sp1/book
        return <book>{
            <isbn>{data($sp2/isbn)}</isbn>,
            <booktitle>{data($sp2/title)}</booktitle>
        }</book>
    }</author>
}</library>
```

(b) Adapted mapping for the addition of chapter$_{ts}$:

```
<library>{
    for $sp1 in S1/library/author
    return <author>{
        <id>{data($sp1/id)}</id>
        for $sp2 in S1/library/author/book
        for $sp3 in S2/library/book
        where $sp2 = $sp3 and $sp2 = $sp1/book
        return <book>{
            <isbn>{data($sp2/isbn)}</isbn>,
            <booktitle>{data($sp2/title)}</booktitle>,
            <chapter></chapter>
        }</book>
    }</author>
}</library>
```

(c) Adapted mapping for the addition of chaptertitle$_{ts}$ and chaptertitle$_{s2}$≅chaptertitle$_{ts}$:

```
<library>{
    for $sp1 in S1/library/author
    return <author>{
        <id>{data($sp1/id)}</id>
        for $sp2 in S1/library/author/book
        for $sp3 in S2/library/book
        where $sp2=$sp3 and $sp2 = $sp1/book
        return <book>{
            <isbn>{data($sp2/isbn)}</isbn>,
            <booktitle>{data($sp2/title)}</booktitle>
            for $sp4 in S2/library/book/chaptertitle
            where $sp3/chaptertitle = $sp4
            return <chapter>
                <chaptertitle>{data($sp4)}</chaptertitle>
            </chapter>
        }</book>
    }</author>
}</library>
```

Figure 4-14. Mapping Adaptation for target element and its correspondences addition

6.2.2. Repairing the Mapping after the Addition of a Mandatory and Monovalued Text Elements

Consider that a text element *n* is added in the target schema such that *n* is mandatory and monovalued. If a mapping contains an assignment for the parent of *n*, the mapping is affected by the addition and needs to be adapted. Given the `FWR` expression *ep* for the parent of *n*, the adaptation consists in extending *ep* to assign *n*.

We distinguish two cases depending on whether the `FWR` expression *ep* already populates some text elements or does not populate any text element. If *ep* already populates some text elements, the adaptation needs to add in *ep* an assignment for *n* from a source element *n'*; *n'* has to be related to the other source elements that are used in a target assignment of *ep*. If *ep* does not populate any target element, the adaptation needs to add in *ep* an assignment for *n*; *n* has to be related to an element used in a target assignment of the `FWR` expression in which *ep* is nested.

If the `FWR` expression already populates some text elements, mapping adaptation follows the same algorithm as for repairing the `return` clause for a source element removal (ref. Section 5.2.1). In both the two situations, the mapping need to assign a target element and to keep it related to the other elements of the `FWR` expression. The only difference is that there is no invalid target assignment that needs to be replaced in the case of a target element addition. Consider that the text element *n* is added in the target schema with some semantic correspondences. The source elements that can be used to assign *n* in the corresponding `FWR` expression *ep* are the ones that satisfy the conditions specified in Section 5.2.1. If a source element *n'* satisfies the condition, it then can be used to assign *n* in *ep*. The mapping adaptation follows the same principle as the one presented in Section 5.2.1; except that only the new target assignment will be added and there is no invalid assignment to be removed. Like in Section 5.2.1, the mapping is not adaptable if no source element satisfies the conditions.

Adapting the mapping if the FWR expression does not populate any text element is similar to the repairing a grouping condition after a source element removal (ref. Section 5.2.6). Consider a monovalued text element *n* added in the target schema and the corresponding FWR expression *ep*. A source element *n'* can be used to assign *n* in *ep* if it satisfies that:

- $n \cong n'$;
- if *ep* is nested in another FWR expression *ep'*: there is an element-binding *for* clause *f'* in *ep'* such that *f'* binds a variable to an element *e'* and either *n* is monovalued with respect to *e'*, or *e'* is monovalued with respect to *n*.

If an element satisfies these two conditions, it is an adaptation solution and the mapping is adaptable. Algorithm 4-5 describes the adaptation in case of a target element addition. It takes as input the new target element *n* and the FWR expression *ep'* in which the FWR expression for *n* is nested. It produces the set of substitutions (*E*) that can be used to replace the grouping condition. For every source element that is equivalent to *n* and every element-binding for clause *f'* in *ep'*, the algorithm checks if the source element is monovalued with respect to the element of *f'* or the inverse.

Algorithm 4-5. The algorithm for finding elements to assign an added target element

```
Elements_Serach_for_a_ Target_Element_Addition(n, ep', E)
Begin
    S := ∅;
    for each element n' such that n ≅ n':
        for each element-binding for clause f' in ep':
            m := binding-element(f);
            if monovalued(m, m') or monovalued(m', m)
                E := E ∪ {n'};
End
```

The mapping is adaptable if some elements that satisfy the conditions are found. For every source element *n'* that satisfies the condition, the adaptation consists in adding the following components in *ep*:

- an element-binding for clause *f* that binds a variable *v* to *n'*;
- a join condition involving either *n'* or *e* related to *f'* and to *f*;
- an assignment from *n'* to *n* in the return clause of *ep*;

6.2.3. Repairing the Mapping after the Addition of a Mandatory and Multivalued Text Elements

If the added target element is multivalued and mandatory, we need to build a new FWR expression for *n* and add it in the FWR expression populating the parent of *n*. As for the addition of a monovalued target element in a FWR expression that does not populate any text element, the adaptation of the mapping has to assure that the instances of *n* are grouped by the instances derived by the FWR expression in which the new expression is nested.

Consider the added multivalued text element *n* and the FWR expression *ep* populating the parent of *n*. A source element *n'* can be used to assign *n* if it satisfies that:

- $n \cong n'$;
- there is an element-binding for clause *f'* in *ep* such that *f'* binds its variable to an element *e'* and either *n* is monovalued with respect to *e'*, or *e'* is monovalued with respect to *n*.

For every source element satisfying the conditions to be an adaptation solution, mapping adaptation consists in defining a `FWR` expression with the following components:

– an element-binding `for` clause *f* that binds a variable *v* to *n'*;
– a `where` clause that contains a grouping condition involving either *n'* or *e* related to *f'* and to *f*;
– a `return` clause with an assignment from *n'* to *n*; *n'* is related to *v*.

Once this `FWR` expression is built, it is nested in the `FWR` expression of the parent of *n* to produce the adapted mapping.

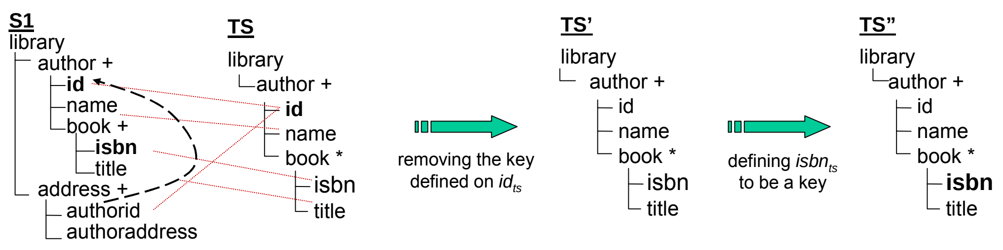
Consider again the example shown in Figure 4-14. We also add another target element *chaptertitle*$_{ts}$ in the target schema (in bold and underlined) with a correspondence *chaptertitle*$_{ts}$ ≅ *chaptertitle*$_{s2}$. The element *chaptertitle*$_{ts}$ is a mandatory and multivalued text element. The mapping of Figure 4-14 (b) is therefore affected and a new FWR expression needs to be built for *chaptertitle*$_{ts}$. The mapping contains the `FWR` expression *ep'* populating *chapter*$_{ts}$. The instances of *chaptertitle*$_{ts}$ must be related to the instances of *ep'*. The element *chaptertitle*$_{s2}$ is a solution: it is equivalent to *chaptertitle*$_{ts}$ and *book*$_{s2}$ is monovalued with respect to it. It is therefore used to adapt the mapping and the result is shown in Figure 4-14 (c). A new `FWR` expression is created for *chapter*$_{ts}$ containing (i) an element-binding `for` clause that binds the variable *$sp4* to the element *chaptertitle*$_{s2}$; (ii) a grouping condition to group the instances of the *chaptertitle*$_{s2}$ by *book*$_{s2}$: *$sp3/chaptertitle = $sp4*, (iii) and a `return` clause that contains an assignment from *chaptertitle*$_{s2}$ to *chaptertitle*$_{ts}$; *chaptertitle*$_{s2}$ is related to *$sp4*.

6.3. Removal of Target Keys

According to our pattern, a mapping may contain a duplicate-elimination `for` clause and some duplicate-elimination statements if an element is defined as a key in the target schema. Consider that the key constraint defined on the element *n* is removed from the target schema. This removal affects the mapping by invalidating all the concerned duplicate-elimination components: (i) the duplicate-elimination `for` clause *f* for *n*, (ii) and the duplicate-elimination statements that involve the variable defined by *f*.

The mapping is always adaptable for a key removal from the target schema. Mapping adaptation consists in removing the invalidated `for` clause and the invalidated statements.

Figure 4-15 shows an example for both target key removal and addition. There is one target schema, one source schema and a mapping defined between them (in Figure 4-15 (a)). We first describe the example for the target removal. The element *id*$_{ts}$ is defined as a key in the target schema, and there is a duplicate-elimination `for` clause and a duplicate-elimination statement for this element in the original mapping (represented in bold and in italic). Consider that the key defined on *id*$_{ts}$ is removed which transforms the target schema from *TS* to *TS'*. The mapping is affected because the duplicate-elimination component is concerned. Mapping adaptation for this change consists in removing all these components; the result is the mapping shown in Figure 4-15 (b).



(a) Original mapping:

```
<library>{
    for $au in distinct-values(S1/library/author/id)
    for $sp1 in S1/library/author[id=$au][1]
    return <author>{
        <id>{data($sp1/id)}</id>,
        <name>{data($sp1/name)}</name>
        for $sp4 in S1/library/author/book
         where $sp1/book = $sp4
        return <book>{
            <isbn>{data($sp4/isbn)}</isbn>,
            <title>{data($sp4/title)}</title>
        }</book>
    }</author>
}</library>
```

(b) Adapted mapping after having removed the key definition over id_{ts}:

```
<library>{
    for $sp1 in S1/library/author
    return <author>{
        <id>{data($sp1/id)}</id>,
        <name>{data($sp1/name)}</name>
        for $sp4 in S1/library/author/book
        where $sp1/book = $sp4
        return <book>{
            <isbn>{data($sp4/isbn)}</isbn>,
            <title>{data($sp4/title)}</title>
        }</book>
    }</author>
}</library>
```

(c) Adapted mapping after having defined $isbn_{ts}$ as a key :

```
<library>{
    for $sp1 in S1/library/author
    return <author>{
        <id>{data($sp1/id)}</id>,
        <name>{data($sp1/name)}</name>
        for $bk in distinct-values(S1/library/author/id)
        for $sp4 in S1/library/author/book[isbn=$bk][1]
        where $sp1/book = $sp4
        return <book>{
            <isbn>{data($sp4/isbn)}</isbn>,
            <title>{data($sp4/title)}</title>
        }</book>
    }</author>
}</library>
```

Figure 4-15. Mapping Adaptation after the removal of a target key and the addition of another target key

6.4. Addition of Target Keys

Adding a target key consists in defining an element of the target schema as a key; the target element has to be mandatory. The addition makes the mapping affected. The mapping adaptation in this case consists in adding a duplicate-elimination `for` clause for the new key and also the necessary duplicate-elimination statements.

Consider that a key is added in the target schema over the element *n* and the original mapping derives instances for *n*. The mapping is therefore affected. A duplicate-elimination `for` clause needs first to be defined and then some duplicate-elimination statements will be added. Given the `FWR` expression *ep* populating *n*, defining the `for` clause requires finding a set of source elements such that element *n'* satisfies the following conditions:

- $n \cong n'$;
- there is an element-binding *for* clause *f* in *ep* such that either *f* binds a variable to *n* or *n* is monovalued with respect to the element to which *f* binds its variable.

The above conditions are similar to the condition for finding elements to re-define an invalid duplicate-elimination `for` clause in Section 5.2.3. As we already mentioned in Section 5.2.3, at least one element will be found following this condition. The element *n* has to be mandatory. Therefore there is at least one source element that is used to assign n in the mapping and that satisfies the above condition.

Given a set of source elements $\{n_1, .., n_k\}$ that satisfies the conditions, the duplicate-elimination `for` clause is defined to bind its variable *$v* to the distinct union of these elements.

A duplicate-elimination statement is created for every source element n_i in $\{n_1, .., n_k\}$: $[n_i = \$v][1]$. Consider the element-binding *for* clause *f* such that that either *f* binds a variable to n_i or n_i is monovalued with respect to the element to which *f* binds its variable. The statement is added in the path expression of the element to which *f* binds its variable.

Consider the example of Figure 4-15. The target schema *TS'* is changed again by defining the element $isbn_{ts}$ as a key which leads to a new target schema *TS”*. The mapping is affected by the change;

a duplicate-elimination `for` clause is firstly defined and then some duplicate-elimination statements are added for the new key. Consider the `FWR` expression *ep* that derives instances of $isbn_{ts}$. There is a source element $isbn_{s1}$ that satisfies the condition to define the duplicate-elimination `for` clause:

– $isbn_{s1} \cong isbn_{ts}$;
– the only element-binding `for` clause of *ep* binds its variable to $book_{s1}$ and $isbn_{s1}$ is monovalued with respect to $book_{s1}$.

A duplicate-elimination `for` clause is added to bind a variable from the distinct instances of $isbn_{s1}$. A duplicate-elimination statement is also added in the concerning `for` clause. The adapted mapping is shown in Figure 4-15 (c).

6.5. Moving of Target Subtrees

As for moving a source subtree, moving a target subtree also consists in removing the old subtree with its constraints and adding the new subtree with the inferred constraints. Since there is no referential constraint in the target schema, the process only infers keys and it follows the same rules as the ones described in Section 5.4.

Moving a target subtree can be considered as a composition of the following changes:

– removing all the target keys that involve an element of *t*;
– removing all the target elements and their correspondences of the subtree in a bottom up manner;
– adding the elements of the subtree *t* at the new place in a top-down manner with the associated correspondences; the cardinalities of elements at the new place are the same as the ones at the original place.
– adding the inferred keys for the elements of the added subtree.

The mapping components invalidated by moving a target subtree are the ones that are invalidated by each change listed above. The mapping is adapted for every composing change following the specific process described in the above sections.

6.6. Renaming of Target Elements

Renaming target elements concerns all leaf or intermediate elements in the target schema and it consists in changing the tag name of the element. The change affects all the components involving this element and the adaptation of the change consists simply in changing the name of this element in the mapping. The mapping is always adaptable in this case.

7. Conclusion

In this chapter, we presented our automatic mapping adaptation approach when schemas evolve. We consider schemas are described in XML Schema and mappings expressed using XQuery; we restrict ourselves to a limited set of XQuery clauses and we define a general patter for the mappings. These mappings are either generated by a specific mapping generation tool or manually specified by a designer.

Our proposal is an incremental approach and it has a specific adaptation process for every change type it considers; each change is propagated in the mappings in three steps: (i) the original mapping is first checked to see if it is affected by the change; (ii) then the affected mapping is checked to see if it is adaptable, which means there is at least one solution to adapt it to the new schemas; (iii) finally adapted mappings are generated if the mapping is adaptable. There may be several adaptation solution; in this case, an adapted mapping is generated for each solution.

Our approach considers the adaptation for both target changes and source changes. It does not rely on a specific mapping generation and it considers the input mappings are either generated by a generation tool or specified manually. It does not require extra metadata except the description of the schemas, the correspondences and the mapping.

Our approach is an incremental approach. For each schema change, we have proposed adaptation algorithm allowing to check if the mapping is adaptable and to build the adapted mapping. These algorithms are the building blocks of a general evolution process; one of our perspectives to define a global propagation process to propagate a set of schema changes. The system may re-order some change propagation or combine some changes to optimize the performance of the adaptation. For this purpose, some priority orders will be attributed to different change types and some algorithms will be developed to recognize the changes that can be combined.

In our approach of mapping adaptation, we define a list of changes that the system considers. Every time one of these changes occurs in the schemas, the system adapts the mapping affected by the change. Few works [MAL05] has been proposed to detect change events. One perspective of our work is to propose an approach and a tool for detecting change events in the schemas. The change events must be detected in real-time. If a new version of a schema is added to the system, the tool should be able to analyse the difference between the two versions and return the set of change events representing the changes. Logs can be used for change detection.

Like our mapping generation approach, our mapping adaptation process also generates a set of adapted mappings. It would be interesting to use some data quality evaluation methods to automatically rank the adapted mappings. The evaluation can be also used inside of the mapping adaptation process to eliminate some intermediate results that will give mappings that are not satisfactory regarding some quality properties.

Chapter 5. Prototype and Experimentation

1. Introduction

Chapters 3 and 4 described our proposal for mapping generation and adaptation for XML schemas. In this chapter we illustrate its practical use in real applications. To this end, we have developed a prototype of an evaluation tool, called AuMGA (Automatic Mapping Generation and Adaptation), which implements the proposed approaches. The tool allows the execution of the algorithms for mapping generation and adaptation as well as some auxiliary functionalities such as the visualization and the edition of the metadata.

In order to validate the approach, the prototype has been used in several application scenarios. Specifically, we describe two data integration applications: the MediaGrid project and an adaptative system for supporting the generation of mediation queries. For each of these applications, we briefly describe its principle and we explain how AuMGA has been used. The validation of our approach in these scenarios is twofold. Firstly, we want to analyze the practical difficulties of modeling different scenarios into AuMGA. Secondly, we want to test the execution of mapping generation method using these applications. We also tested the communication of the tool with the other modules in these applications. We have performed some tests for evaluating the performance of the mapping generation approach. For this purpose, we first consider some typical test scenarios and we execute our method over every scenario. Secondly, we evaluate the performance of the different steps of the approach.

The following sections describe our experimentations: Section 2 presents the AuMGA tool, describing its functionalities, architecture and its user interface. Section 3 describes the experience of using AuMGA in the two application scenarios. Section 4 presents the performance evaluation tests, describing the considered test scenarios, the tests strategies and the obtained results. Finally, Section 5 concludes the chapter.

2. Prototype

We developed a prototype system to implement the approach of mapping generation and adaptation and it is called AuMGA. The tool is implemented in Java (JDK 5.0) using the external package Apache Xerces [xerces]. The source of the prototype implementation has been made available at SourceForge.net[2] under the GNU General Public License (GPL).

In the remainder of the chapter, Section 2.1 first describes the functionalities provided by the system. Section 2.2 presents the system architecture and Section 2.3 describes the user interfaces provided by the tool.

[2] The AuMGA project at SourceForge.net: https://sourceforge.net/projects/aumga

2.1. System Functionalities

The system implements the functionalities required for mapping generation and mapping adaptation. These functionalities can be classified into the following categories:

- Mapping generation functionalities;
- Mapping adaptation functionalities ;
- Metadata management functionalities
- Visualization and editing functionalities
- Application management functionalities

In the rest of the section, we will describe each of these categories.

2.1.1. Mapping Generation Functionalities

The system can generate a set of mappings for the given schemas. The approach takes as input one target schema and several source schemas, all described in XML Schema. It also takes as input a set of semantic correspondences between elements of different schemas. It generates several mappings for the target schema from the source schemas; each mapping represents alternative semantics to derive instances of the target schema. The generated mappings are described in an abstract representation and the system can also translate these abstract mappings into XQuery. To support this process, the tool includes the following functionalities:

- target schema decomposition: produces a set of target subtrees given a target schema;
- source parts identification: produces the relevant parts of the sources for a given target subtree;
- candidate join identification: produces the set of candidate joins given a set of source parts for a given target subtree;
- partial mapping definition: produces the set of partial mappings for a given target subtree given the corresponding source parts and candidate joins;
- target mapping generation: produces the abstract target mappings by considering all the valid combinations of partial mappings;
- XQuery translation: produces the XQuery expression of an abstract mapping;
- Incomplete results generation: produces a new target schema by relaxing some constraints; this schema can be used as input to restart the generation process.

2.1.2. Mapping Adaptation Functionalities

The process of mapping adaptation modifies the mapping after a source or a target schema change. It considers a mapping between one target schema and several source schemas; the process generates, if possible, several adapted mappings for the new schemas. The tool includes the following mapping adaptation functionalities:

- testing if a mapping is affected: checks the different components of a mapping to see if some of them are invalid;
- testing mapping adaptability: produces the different adaptation solutions.
- repairing an affected mapping: produces a new mapping from an affected mapping;.

2.1.3. Metadata management Functionalities

The tool also proposes several auxiliary functionalities to allow the management of the data and of the different results of mapping generation and adaptation.

The system gets input and it also outputs information as execution results. The system allows users to input the schemas, the semantic correspondences and the mappings. The input data can be from files: the schemas are defined in XML Schema [xsd] files (.xsd files), the correspondences are defined in XML files, and the mappings are expressed in XQuery files (.xq files). The files are read into the system using specific parsers. The same information can be also retrieved in a meta-data server following a predefined model. The system communicates with the server through JDBC [jdbc].

Every time the system takes some information as input, it checks if the information is valid and translated into the internal representation of the system. For example, nodes related by a semantic correspondence have to exist in the schemas.

The system also outputs information. It produces generated mappings or adapted mappings, as well as new target schemas after constraint relaxation. The system can also output the evolved schemas and their related semantic correspondences. The data may be outputted as files or be transferred to the data server.

2.1.4. Visualization and Editing Functionalities

Users need to visualize the input data as well as the intermediate and final results of the executions. The system allows the visualization of the schemas, the semantic correspondences and the mappings. Users sometimes can choose between different visualization options. As an example, the schemas can be shown in a graphical representation or in a textual representation: the graphical representation is more intuitive while the textual representation is more detailed with all the properties of the schemas.

The system also allows the visualization of the intermediate results during the execution of a process. The user interface is automatically refreshed every time after the execution of a step.

An editor of metadata allows users to edit the schemas and the semantic correspondences. This allows the user to build a target schema and to modify some existing schemas. The system checks for every change its validity before performing it (e.g. an added reference should reference to an existing key in the schema).

2.1.5. Application management functionalities

Beside the above functionalities, the adaptation and generation tool provides some additional functionalities such as the management of sessions. A session represents a concrete application scenario and encloses the components for that scenario, i.e. schemas, correspondences, partial mappings. Several sessions can be stored but the tool only manages one session at a time. Among the functionalities of management of sessions, the tool includes: the creation, loading, storing and deletion of sessions.

2.2. System Architecture

Figure 3-1 shows the architecture of our system. There are mainly three layers: data storage, logic and view.

The ***data storage layer*** manages data including schemas, correspondences and mappings. It communicates with the logic layer to provide input information and to obtain result data. The input data can be retrieved from a meta-data server or from files. The output of the system can be generated mappings, adapted mappings and also changed schemas and correspondences.

The ***logic layer*** contains the implementation of the processes of: mapping generation, incomplete result generation and mapping adaptation. This layer communicates with the data storage layer to take data as input. Different modules communicate via the data storage layer. As an example, the mapping generation modules produce mappings and output them to the data storage layer. The system checks if there is a result mapping after every execution of the mapping generation and it decides or not to execute the process of incomplete result generation. Later on, the mapping adaptation module can also take as input from the data storage layer the mapping produced by the other modules.

The ***view layer*** contains all the utilities to allow the logical layer to communicate with the user. Through the view layer, the user can explore the input data (i.e. schemas, etc.) and monitor the intermediate and final results of the executions. It allows the user to visualize a relaxation option, to validate a relaxation as well as to choose between different options. It also allows make some changes into the schemas or into the set of correspondence; the latter will trigger the execution of the mapping adaptation if some mappings are established between the target schema and the source schemas of the system.

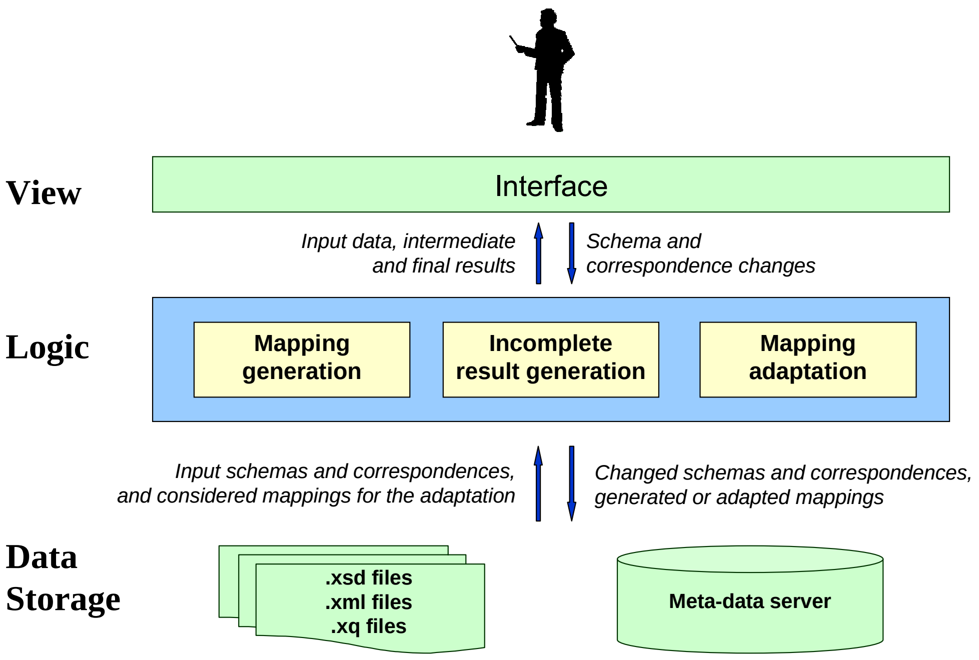

View
Interface
Input data, intermediate and final results
Schema and correspondence changes
Logic
Mapping generation
Incomplete result generation
Mapping adaptation
Input schemas and correspondences, and considered mappings for the adaptation
Changed schemas and correspondences, generated or adapted mappings
Data Storage
.xsd files
.xml files
.xq files
Meta-data server

Figure 5-1. The architecture of the mapping generation system

2.3. User Interfaces

In this section, we present the graphical user interfaces (GUI) of AuMGA through two screen snapshots: one showing a panel of schemas and correspondences and the other showing a panel of mapping generation.

2.3.1. Schemas and Correspondences

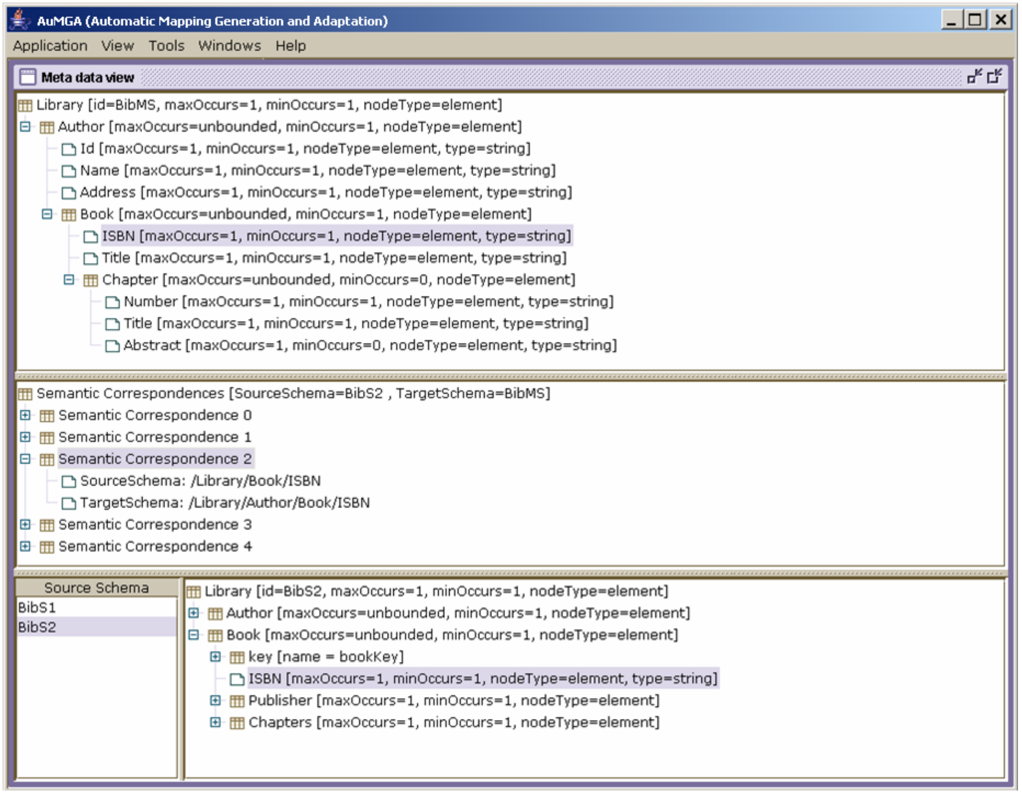
Figure 5-2 shows a user interface in AuMGA to visualize the schemas and correspondences of an application session. The main window contains a main menu on the top. This menu allows starting, closing and saving a session; allows starting mapping generation or mapping adaptation; and allows personalizing looks and feeling of the application.

The interface of Figure 5-2 shows the schemas and the correspondences used by a session. For a given session, this interface allows users exploring all the schemas and the correspondences used in the session. The user can edit the schemas and the correspondences through the interface. As we can see in the snapshot, the interface of meta-data contains three panels in vertical:

– ***the panel of the target schema***: it is the panel on the top of the interface. The target schema is visualized in a text representation. We can see that the elements of the schema are organized by their hierarchical structure. Users can see the whole schema and it can also hide some subtrees of the schema. Using this interface, users can edit the target schema, such as adding or removing a target element.

– ***the panel of source schemas***: this panel is shown at the bottom of the interface. At the left side of the panel, there is a list of the source schemas that are considered in the session. Users can select one schema in the list and the content of the schema will appear at the right side of the panel. Here, users can edit the set of source schemas; such as removing an element in a source schema, adding a source schema and removing a source schema.

– ***the panel of semantic correspondences***: this is the panel at the middle of the interface. It gives all the semantic correspondence between the target schema on the top and the selected source schema shown at the bottom. Users can also edit these semantic correspondences such as removing some correspondences or adding some new correspondences.


AuMGA (Automatic Mapping Generation and Adaptation)
Application View Tools Windows Help
Meta data view
Source Schema

```
Library [id=BibMS, maxOccurs=1, minOccurs=1, nodeType=element]
  Author [maxOccurs=unbounded, minOccurs=1, nodeType=element]
    Id [maxOccurs=1, minOccurs=1, nodeType=element, type=string]
    Name [maxOccurs=1, minOccurs=1, nodeType=element, type=string]
    Address [maxOccurs=1, minOccurs=1, nodeType=element, type=string]
    Book [maxOccurs=unbounded, minOccurs=1, nodeType=element]
      ISBN [maxOccurs=1, minOccurs=1, nodeType=element, type=string]
      Title [maxOccurs=1, minOccurs=1, nodeType=element, type=string]
      Chapter [maxOccurs=unbounded, minOccurs=0, nodeType=element]
        Number [maxOccurs=1, minOccurs=1, nodeType=element, type=string]
        Title [maxOccurs=1, minOccurs=1, nodeType=element, type=string]
        Abstract [maxOccurs=1, minOccurs=0, nodeType=element, type=string]

Semantic Correspondences [SourceSchema=BibS2 , TargetSchema=BibMS]
  Semantic Correspondence 0
  Semantic Correspondence 1
  Semantic Correspondence 2
    SourceSchema: /Library/Book/ISBN
    TargetSchema: /Library/Author/Book/ISBN
  Semantic Correspondence 3
  Semantic Correspondence 4

BibS1
BibS2

Library [id=BibS2, maxOccurs=1, minOccurs=1, nodeType=element]
  Author [maxOccurs=unbounded, minOccurs=1, nodeType=element]
  Book [maxOccurs=unbounded, minOccurs=1, nodeType=element]
    key [name = bookKey]
    ISBN [maxOccurs=1, minOccurs=1, nodeType=element, type=string]
    Publisher [maxOccurs=1, minOccurs=1, nodeType=element]
    Chapters [maxOccurs=1, minOccurs=1, nodeType=element]
```

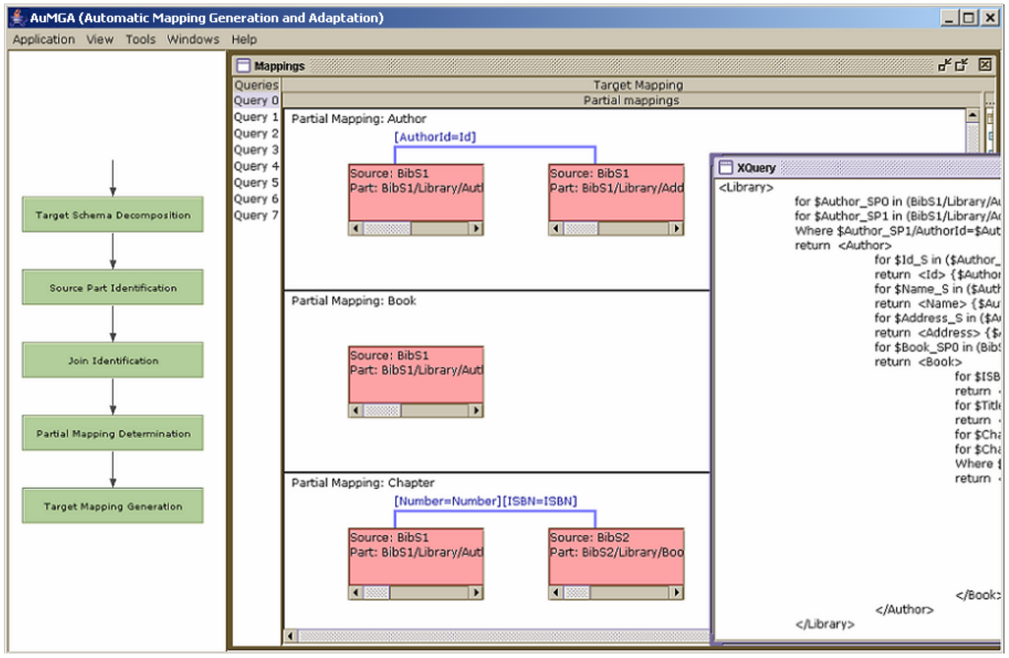
Figure 5-2. Graphical user interface of Meta-data in AuMGA

2.3.2. Mapping Generation

Figure 5-3 shows an interface of mapping generation. The snapshot shows some generated mappings in an application session. It contains two main components:

– ***the panel of mapping generation***: this panel is shown at left of the interface. It gives the steps of the mapping generation. On the panel, every step can be active or inactive. An inactive step can not be executed and user can click on the active steps to execute them. A step becomes active only if its previous steps have been executed. In Figure 5-3, all the steps are active.

– ***the panel of result***: Apart from the panel of mapping generation, the rest of the interface is used to visualize the intermediate or final results. The snapshot in Figure 5-3 shows the generated target mappings for the input schemas and correspondences of Figure 5-2. There is a list of 8 mappings generated form the schemas. The list is shown right next to the the panel of mapping generation procedure. Among the generated mappings, the select mapping (highlighted) is shown at the right side of the list: the target subtrees that are defined by the mapping using which partial mappings. These mappings expressed as a set of partial mappings. They can also be translated into XQuery; at the very right side, an XQuery query is given for the selected mapping.



```
<Library>
        for $Author_SP0 in (BibS1/Library/Au
        for $Author_SP1 in (BibS1/Library/Ad
        Where $Author_SP1/AuthorId=$Aut
        return  <Author>
                for $Id_S in ($Author_
                return  <Id> {$Author
                for $Name_S in ($Auth
                return  <Name> {$Au
                for $Address_S in ($A
                return  <Address> {$
                for $Book_SP0 in (BibS
                return  <Book>
                        for $ISB
                        return ·
                        for $Titl
                        return ·
                        for $Cha
                        for $Cha
                        Where $
                        return ·



                        </Book>
                </Author>
        </Library>
```

Figure 5-3. Graphical user interface of generated mappings in AuMGA

3. Applications

In order to validate the approach, the prototype has been used in several application scenarios. We describe two data integration applications: the MediaGrid project and an adaptive system for supporting the generation of mediation queries. For each of these applications, we briefly describe its principle and we explain how AuMGA has been used.

3.1. The MediaGrid Project[3]

The MediaGrid is a multidisciplinary project supported by the French ministry of research through the ACI-Grid program[4]. The MediaGrid project proposes a mediation framework for a transparent access to data sources in a distributed and dynamic environment.

[3] The MediaGrid Project, a mediation framework for a transparent access to data sources: http://www-lsr.imag.fr/mediagrid/

The project contains a mediation query generator to generate automatically the mediation queries between the mediation schema and the exported schemas of the sources. The mediation query generator is built using our tool.

The MediaGrid framework considers both the Local-as-View (LaV) approach and the Global-as-View (GaV) approach for data integration. The mediation query generator has to therefore generate mediation queries in the two kinds of scenarios. The generations of mediation queries in both of the two kinds are supported by AuMGA. In a GaV approach, generating the mediation queries consists in generating mappings for the mediation schema from the exported schemas. In a LaV approach, generating the mediation queries consists in generating mappings for every exported schema from the mediation schema.

The mediation query generator is validated using the biological domain application of the project (more characteristics will be given in the performance evaluation in Section 0). The application consists in three biological sources GOLD, SGD and SMD. A wrapper is built for every of these sources and it generates an exported schema from the source and export it to the mediation system; the schemas are expressed in XML Schema. A mediation schema is designed by some biologists and semantic correspondences are given between the elements of the mediation schema and their equivalents elements in the three sources. From the input information, the mediation query generator generates one mediation query in a GaV approach that defines the mediation schema from the three sources. This mediation query is validated by the biologist that it integrates the three sources in a right way and represents a meaningful semantics to define the target.

3.2. An Adaptive System for Supporting the Mediation Query Generation

In [KPS05], we presented an adaptive system for aiding the mediation query generation. The goal of the system is to generate several mediation queries, evaluate the quality of their data and select the most appropriate according to user preferences.

A design toolkit provides these functionalities. Figure 5-4 shows the architecture of the toolkit. It is composed of three tools: ***mediation query generation***, ***user profile management*** and ***data quality evaluation***. The tools communicate via a server of meta-data which store all the metadata used and produced by the tools. A definition interface allows users to interact with the tools. The mediation query generation tool is charged to generate mediation queries. The user profile management tool [Kos06] is responsible for the definition of user profiles, which include quality expected values. The data quality evaluation tool [Per06] is responsible for the estimation of the data quality provided by the generated queries and the selection of those satisfying user quality expectations.

[4] ACI GRID (Actions Concertées Incitative - Globalisation des Ressources Informatiques et des Données): http://www-sop.inria.fr/aci/grid/public/acigrid.htm

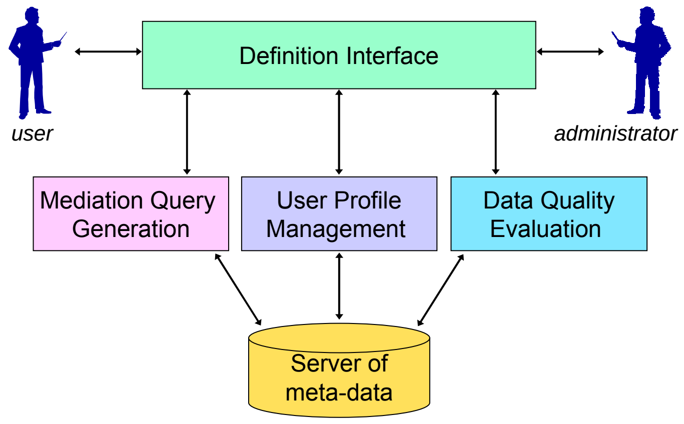


Figure 5-4. The toolkit architecture

The mediation query generation tool uses AuMGA to generate possible mediation queries between the global schema and the source schemas. Since the data quality provided by a query also depends on users expectations, it is necessary that the tool generates all the possible mediation queries by considering all the sources and all the combination of the sources. Then data query evaluation tool will evaluate the qualities of the query answer to allow selecting the desirable ones for the user.

4. Performance Evaluation

We have used five scenarios ranging from a simple scenario of a target schema defined over 3 data sources to a complex one involving 50 sources. Table 5-1 summarizes the main characteristics of these scenarios, such as the number of elements in the target schema, the number of data sources, the number of elements for each one, the number of correspondences and the number of key definitions in the sources. The first scenario is from the Mediagrid project. The Library1 scenario contains six source schemas. The Library2 scenario is similar to Library1 but the overlap between the sources is more important (more correspondences are defined for the same number of text elements in the target schema). The ABC1 and ABC2 scenarios have 50 source schemas. The only difference between them is that the ABC2 scenario contains 58 key definitions while ABC1 contain no key definitions.

Table 5-1. Characterizing the different scenarios

Scenarios	Target schema			Correspondences	Source schemas				
	Depth	Elements	Text elements		Schemas	Elements	Text elements	Keys	Refs
Mediagrid	6	18	12	22	3	1674	825	412	413
Library1	5	18	14	26	6	56	30	9	1
Library2	5	18	14	30	6	62	35	10	1
ABC1	7	47	36	1300	50	1650	1434	0	0
ABC2	7	47	36	1300	50	1650	1434	58	0

4.1. Evaluating the Different Steps of the Scenarios

We have run these different scenarios on a PC-compatible machine, with a 2.8G Hz P4 CPU and 516MB RAM, running Windows XP and JRE1.4.1. Each experiment is repeated five times and the average of the five is used as the measurement. The time needed for the main steps of our approach using the different scenarios are shown in Table 5-2.

Table 5-2. Measuring the main steps of our approach

Scenarios	Execution time (s)			
	Load	Target Schema Decomposition	Partial Mapping Definition	Target Mapping Generation
Mediagrid	1.44	0.001	0.02	0.002
Library1	0.44	0.001	0.067	0.095
Library2	0.046	0.001	0.105	0.25
ABC1	0.98	0.001	2,844	0,375
ABC2	1.03	0.001	316	27

The loading time indicates the time to read the schemas and the correspondences into our internal representation. As expected, it is correlated to the size of the schemas and the number of their correspondences. The target schema decomposition time indicates the time to decompose the target schema into target subtrees. We can see that the time needed to perform this task is negligible.

The partial mapping definition (pmd) time is proportional to the number of correspondences and the number of key and key references in the sources. The Library1 and Library2 scenarios have the same number of sources and the same target schema, but the pmd time for the Library1 scenario is smaller than the one of the Library2 scenario because the Library2 has more correspondences than Library1. The pmd time for the ABC2 scenario which has 58 keys is largely greater than the one of the ABC1 scenario. This is because the number of keys of the ABC2 scenario makes the join graph very complex.

The target mapping generation (tmg) time indicates the time to find all the candidate mappings and to generate the target mappings. The tmg time is greater in ABC2 than in the other scenarios because it has in average 150 partial mappings per subtree, which leads to much more combinations to consider. From these measures, we can notice that the time required for defining the partial mappings and generating the target mappings is important compared to the time needed for the other tasks, except for the Mediagrid scenario in which there are few correspondences; the solution space is therefore quite little although the source schemas are very big.

The pmd time is decomposed into source part identification time, join identification time and sub-graphs definition time. These times are shown in Table 5-3.

Table 5-3. Measuring the activities of partial mapping definition

Scenarios	Target subtree	Total src parts	Total joins	Src parts identification time (s)	Joins identification time (s)	Sub-graphs definition time (s)
Mediagrid	3	6	5	0,002	0,006	0,012
Library1	3	14	19	0,005	0,012	0,05

Library2	3	15	22	0,006	0,014	0,085
ABC1	5	250	0	0,02	0,034	1,79
ABC2	5	100	120	0,02	5,346	310,2

The task that most contributes to the pmd time is sub-graph definition, which takes 75%, 86% and 99% for the Library1, ABC1 and ABC2 scenarios respectively.

The join identification time increases when the number of source parts and the number of keys in the source schemas increases.

The sub-graph definition consists in enumerating all the paths in the join graphs of every target subtree. The duration of this task depends on the size of the different join graphs. In ABC2, the size of some join graphs is very important (one of them has 50 elements and 37 edges), and the sub-graphs definition time is greater than the one of the other scenarios, in which the size of the join graphs is smaller.

In the following sections, we measure the performances of the different steps of mapping generation with respect to the parameters that influence each of the steps.

4.2. Performance Evaluation of Different Steps of the Mapping Generation Approach

The section gives the results of the performance evaluation over the different steps of mapping generation. All of them use a scenario presented above and vary some critical inputs to measure the performance of the approach.

4.2.1. Decomposition of the Target Schema

The time to perform the target schema decomposition is influenced by the size of the target schema. Figure 5-5 shows the time with respect to the number of elements in the target schema. As we can see, the variation of the target schema decomposition time is negligible.

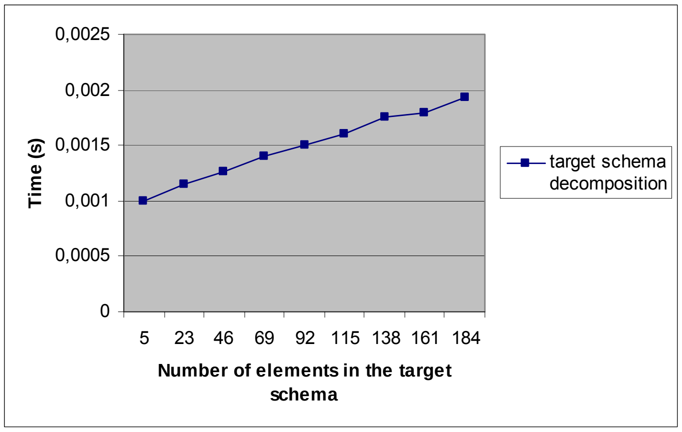


Figure 5-5. Measuring target schema decomposition time

4.2.2. Source Parts Identification

This task is influenced by the number of correspondences between the target schema and the sources because the corresponding algorithm consists in browsing the set of correspondences to identify contributive source elements.

Figure 5-6 shows variation of the time to perform the source part identification with respect to the number of semantic correspondences using the ABC2 scenario. As expected, it is proportional to the number of correspondences and the time to perform the task is almost negligible (about only 22 ms for 1300 correspondences).

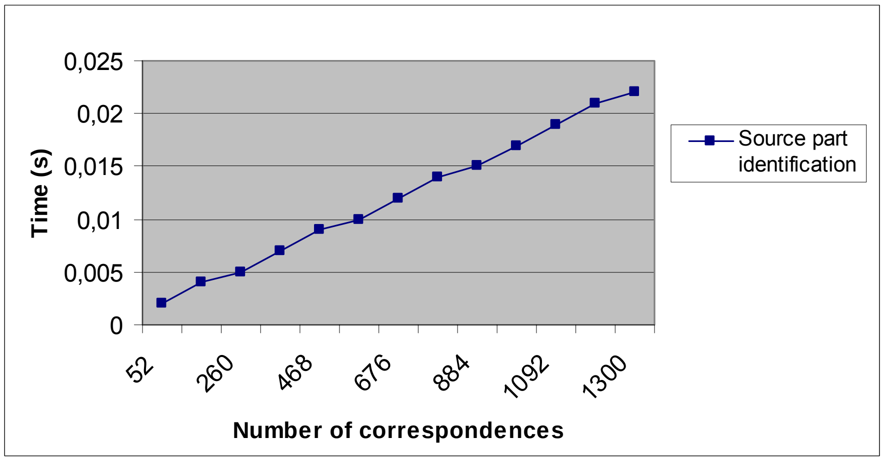


Figure 5-6. Measuring source part identification time

4.2.3. Join Identification

If c and c’ are two sets of source elements belonging to two distinct sources respective, we consider that a join with the predicate c = c’ is possible between these two sources if: (i) the two sets are involved in a correspondence (c ≅ c’) and (ii) either c or c’ is defined as a key. The join identification is therefore influenced by the number of key definitions in the sources and the number of correspondences involving keys. We have conducted three experiments using the ABC2 scenario.

In the first experiment, we keep the same target schema, the same elements in the source schemas and the same correspondences, and we progressively increase the number of key definitions that are involved in a correspondence. Figure 5-7 shows the time of join identification with respect to the number of key definitions in the source schemas.

The process of the join identification is decomposed as follows: (i) it firstly finds a pair of sets c and c’ that satisfies the conditions to make c=c’ being the predicate of the join; (ii) then it searches, for this predicate c = c’, two source parts p and p’ that respectively intersect with c and c’. Finding a pair c and c’ such that c = c’ is a join predicate requires that either c or c’ is a key. Suppose c is a key; adding a key definition on c’ doesn’t lead to new join predicate. At a certain point after having increased the number of keys, if each correspondence involves one key definition, adding new keys doesn’t lead to a new join predicate. We can see that in Figure 5-7, when the number of key definitions is near 200, adding a new key doesn’t influence significantly the time needed for the task.

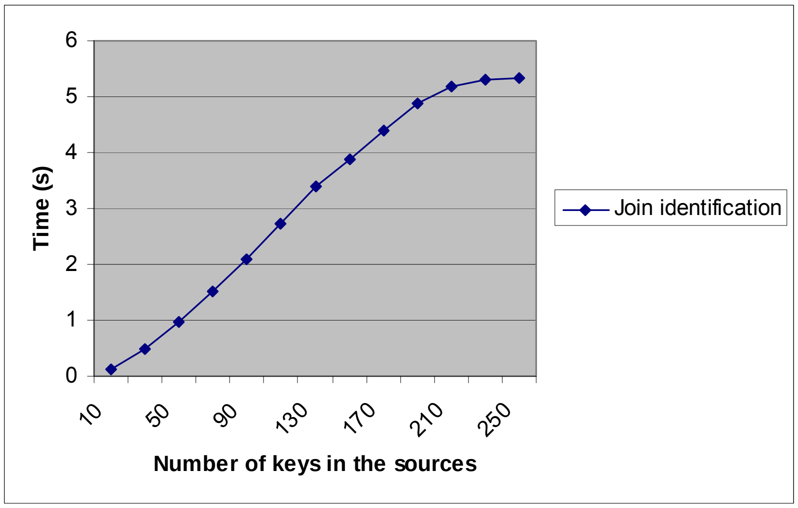


Figure 5-7. Measuring join identification time with respect to the number of key definitions in the sources

In the second experiment represented by Figure 5-8, we kept the same target schema, the same source schemas and the same key definitions. We increased progressively the number of correspondences involving keys to measure the time needed for the join identification process.

In Figure 5-8, we can see that the process increases with respect to the number of correspondences involving keys. Suppose that n is an element of the target schema and k1,...kn+1 a set of elements in the sources, each one defined as a key; suppose also that the correspondences n≅ki, i=1...n. If we add the correspondence k≅kn+1, this will also result in adding correspondences between k and all the elements k1 to kn, which represent n more combinations to explore for finding candidate joins.

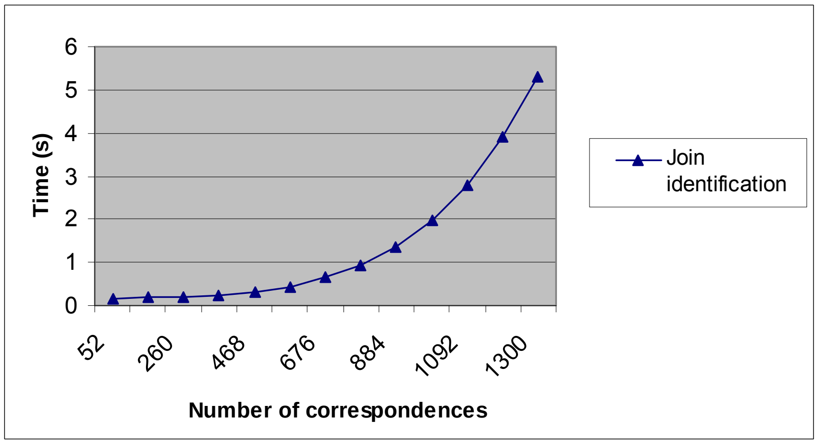


Figure 5-8. Measuring join identification time with respect to the number of correspondences involving a key

The last experiment shown in Figure 5-9 consists in increasing at the same time the two parameters used in the two previous experiments: the number of key definitions and the number of correspondences involving keys. As we can see, with 300 key definitions and 300 correspondences involving keys (which represents a quite complex case), the time for join identification is about 15 seconds.

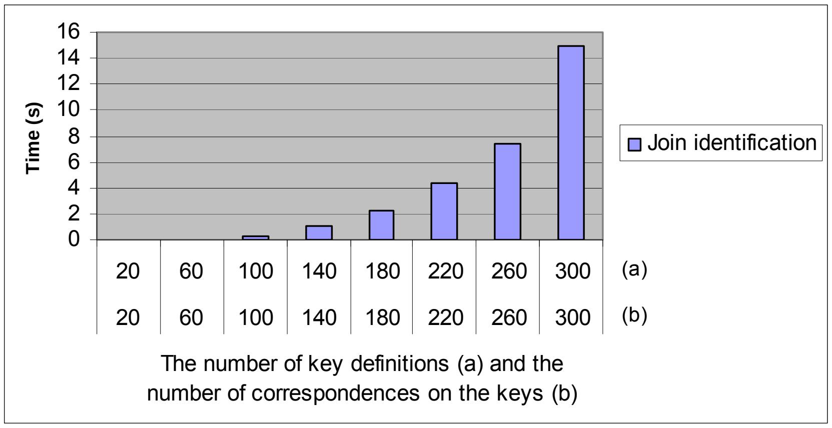


Figure 5-9. Measuring join identification time with respect to both the number of key definitions and the number of correspondences involving keys

4.2.4. Partial Mapping Definition

Defining the partial mappings for a given target subtree consists in enumerating the sub-graphs of the join graph. The time required for this task depends not only on the size of the corresponding join graph (the number of edges and elements), but also on its structure. For example, consider two join graph having the N elements, the one having no edge takes less time to be scanned than the one having N^2 edges. We characterize the structure of the graph using the maximum number of neighbors for an element in the graph. We measure the performance of partial mapping definition with respect to both the size of the corresponding join graph and its structure.

The time for partial mapping definition with respect to the size of the join graph is shown in Figure 5-10 and it is performed using the ABC1 scenario. Keeping the same number of correspondences for each key, the graph shows the time needed for the task with respect to the number of correspondences between the target schema and the source schemas. Increasing the number of correspondences is done by appending new source schemas. We can see that the time increases with the number of correspondences.

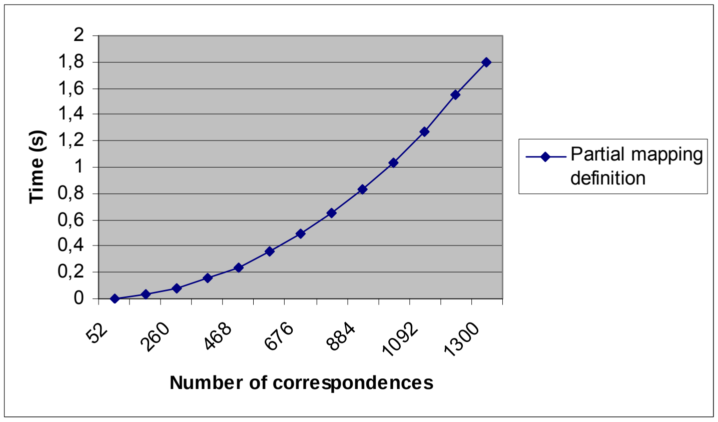


Figure 5-10. Measuring partial mapping definition time with respect to the number of correspondences

The second experiment is represented by Figure 5-11. The experience considers the source schemas and the target schema of the ABC1 and ABC2 scenarios. We select a random key in the scenario initial. By increasing correspondences relating this chosen key, the graph shows the time for partial mapping definition with respect to the number of edges in a graph (the number of the elements doesn't vary). We can see that the time needed for partial mapping definition is exponential with respect to the correspondences number for keys.

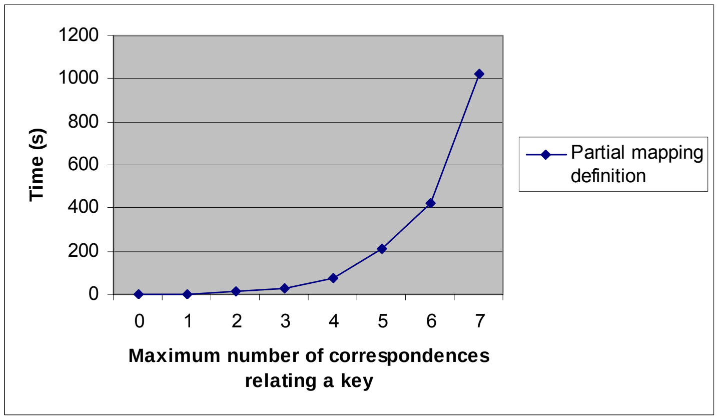


Figure 5-11. Measuring partial mapping definition time with respect to the number of correspondences relating a given key

The experiment in Figure 5-12 shows the time needed for partial mapping definition with respect to both the number of correspondences and the number of correspondences for a given key. The source schemas and the target schema of the scenarios are the same as the Library2 scenario. The graph shows that the task takes about 10 seconds to determine the partial mappings for a scenario having 364 correspondences and 7 correspondences for a key, and only 2 seconds for a scenario having 1300 correspondences and 2 correspondences for a key.

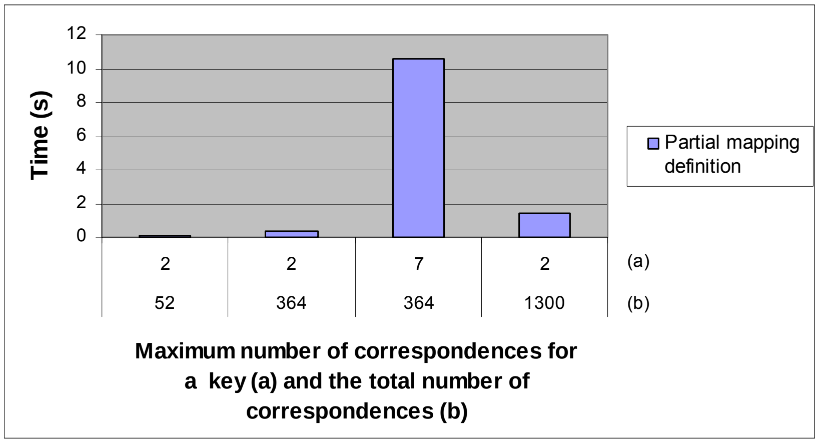


Figure 5-12. Measuring partial mapping definition time with respect to both the maximum number of correspondences for a key and the total number of correspondences

4.2.5. Target Mapping Generation

The time for target mappings generation depends on the number of partial mappings and the structure of the target schema, that is, the number of parent/child relations between target subtrees. Figure 5-13 shows the time required for this task with respect to the number of target subtrees and the average correspondences per subtree. We have increased both parameters using the same sources as for the ABC2 scenario. For example, in the case of 5 target subtrees and 230 correspondences per subtree, it takes about 12 seconds for generating the target mappings; note that this case is a complex one, since the scenario contains 50 sources, 1300 correspondences and 58 key definitions. The complexity of this process is exponential with respect to the number of partial mappings. It is possible to reduce this complexity using some quality criteria (for example, selecting the partial mappings that use the sources having a high confidence factor) or some heuristics (for example, selecting the partial mappings using a high number of sources).

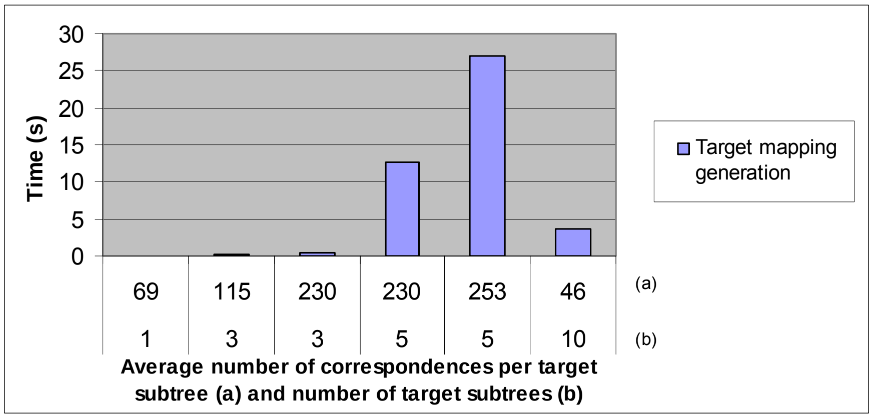


Figure 5-13. Measuring mapping generation time with respect to both the number of target subtrees and the average correspondences per subtree

5. Conclusions

In this chapter we presented our experimentations with mapping generation and adaptation. We described a prototype of mapping generation and adaptation tool, AuMGA, which manages the proposed approaches in the Chapters 3 and 4. We presented the main functionalities, the system architecture and some interfaces.

We used the AuMGA tool in several application scenarios in order to validate our approach. Specifically, we described two data integration applications: the MediaGrid project and an adaptive system for aiding in the generation of mediation queries. We showed how the data integration applications were modeled in AuMGA and how the approaches execute for these specific scenarios. This experimentation allows validating the approach in real applications; especially, it affirms that it is ease to model these data integration systems into AuMGA.

We also describe some tests for evaluating performance. We first consider some typical test scenarios and we execute our method over every scenario and compare the performances of comparable scenarios. We also evaluate the performance over the different steps of the approach using a scenario presented above and vary some critical inputs. The test results affirm that AuMGA can be used for real applications.

The development of the system is an on-going work. We implemented all the algorithms that we described in Chapters 3 and 4 and we also implemented other auxiliary functionalities such as the data management and the visualization of the result. We still need to provide a user interface allowing user to monitor the execution of the mapping adaptation. We also plan on doing some evaluation tests over the algorithms of mapping adaptation to measure its performance.

Chapter 6. Conclusions

1. Summary of the Contributions

This thesis addresses the problem of automatic mapping generation and adaptation. We consider one target schema and several source schemas, all described in XML Schema. We generate mappings to derive instances of a target schema from the instances of the set of source schemas.. We also adapt mappings when the target schema or one of the source schemas evolves. A set of semantic correspondences are used to perform these tasks.

To handle the complexity of mapping generation, our approach comprises three steps:

– decomposing the target schema into a set of target subtrees;
– then finding partial mappings to derive instances of each target subtree from the instances of the source schemas; the partial mapping generation for each target subtree is done independently from the others.
– finally combining the partial mappings of different target subtrees to generate the target mappings for the whole schema.

The result of the mapping generation is a set of mappings; every one represents an alternative way to derive instances of the target schema. The result mappings are specified in an abstract language and can be translated into a specific language such as XQuery.

If the target schema can not be satisfied by the sources, no mapping can be generated. We proposed an adaptation of our generation process to provide incomplete results if the data sources do not allow generating a mapping satisfying the users' needs as represented by the target schema. This is done by relaxing some cardinality constraints or some structural constraints of the target schema such that a mapping can be found for the new schema. The process is an interactive process in which the user is asked to validate the modifications made on the target schema and to choose one relaxation option among the other alternatives.

A mapping it may become obsolete when the target schema or one of the source schemas evolves. This thesis proposes a solution that adapts a mapping according to the changes to generate mappings for the new schemas. We consider a set of changes and we specify an adaptation process for every of these changes in three steps:

– first checking the original mapping to see if it is affected by the change.
– if the mapping is affected, checking if it is adaptable, which means there is at least one solution to adapt it to the new schemas.
– if the mapping is adaptable, mappings are generated for the new schemas following all the adaptation solutions found during the second step.

The result of this approach is a set of adapted mappings; every one represents an alternative way to adapt the mapping to the new schemas.

Compared to the existing works, our generation approach produces mappings from multiple source schemas and it generates mappings involving joins between different sources. It does not assume that the target schema and the source schemas have homogeneous structures. It generates mappings expressed in an abstract format such that they can be translated into any specific language. It can also translate these mappings into XQuery. Our adaptation approach considers a large amount of change types and it enables the adaptation for both target changes and source changes. The approach does not rely on a specific mapping generation and it considers the input mappings are either generated by a generation tool or specified manually. It does not require extra information except for the schemas, the correspondences and the mapping.

The approach of automatic mapping generation led to results of several publications. In [KX05b] we have proposed a general framework for mapping generation; in [KX05c, KX05d], we have presented the generation approach, some of our algorithms and some experimental results; and in [KX05a], we have presented the complete set of algorithms for generating abstract mappings and the translation process to XQuery; we have also extend the approach to take into account 1-n correspondences.

We developed a prototype implementation of the system, called AuMGA. It implements all the algorithms presented in the thesis. We used this prototype to validate our approaches in different application scenarios and to evaluate the performances. The source of the implementation has been made available at SourceForge.net as open source under the GNU General Public License (GPL). The prototype of the mapping generation approach has been demonstrated in the conference Bases de Données Avancées (BDA) 2004 [KPS04]. We still need to develop the interface to allow the user to monitor the execution of the mapping adaptation process. We also plan on doing some evaluation tests over the algorithms of mapping adaptation to measure its performance.

The result of our mapping generation and adaptation processes depend on the input metadata: if the set of available metadata is complete, more possibilities are found to generate mappings that conform to the user's expectation. As an example, if the schemas do not contain any constraints allowing the element identification, few mappings will be generated.

The approach of mapping generation enumerates all the possible solutions to generate mappings and produces a set of mappings that represent different ways to define the target from the sources. The process therefore may become inefficient in case there are many equivalent elements between different source schemas and many constraints defined on these elements. The process needs to enumerate all the possible combinations between the sources; the complexity of the process is exponential. Too many mappings may also be generated to make the user choice difficult. Some heuristics or data quality criteria may be used to reduce this complexity.

2. Outlooks

In this thesis, we have addressed the problem of mapping generation and adaptation; we describe below different research perspectives that can be drawn. Some of them consist in developing tools to satisfy the assumption of the methods proposed in the thesis: the generation of semantic correspondences, the detection of schema and correspondence changes. Some methods consist in optimizing the existing approaches: the global propagation of changes in mappings and the cost driven method to choice between mapping generation and mapping adaptation for change adaptation. The others consist in adapting the existing methods in new contexts: the adaptation of the approaches in peer-to-peer architecture and the adaptation of the approaches in contexts supporting quality evaluations.

2.1. Semantic Correspondence Generation

In our approach, we consider that a set of semantic correspondences is provided; they can be 1-1 correspondences or 1-n correspondences involving transformation functions. The generation of these semantic correspondences is called schema matching.

Many approaches [DLD04, DR02, Gal06, HC06, RB01, RSV01, and SE05] have been proposed to generate semantic correspondences between schemas. However most of them still mainly focus on 1-1 correspondence generation. Only two approaches allow generating more complex 1-n correspondences: iMap [DLD04] relies on the availability of instance values to construct complex correspondences from simple 1-1 correspondences; the DCM framework proposed in [HC06] consists in a correlation mining approach to explore the co-occurrence information across Web query interfaces. However, the 1-n semantic correspondences that can be found by these approaches are still limited.

One of our perspectives is therefore to provide a tool to support the automatic generation of 1-n correspondences. There exist different methodologies for discovery correspondences. Some approaches infer correspondences based on the schema constraints (keys and references); while some others make use of a linguistic thesaurus to establish relationships between different concepts. Some approaches consider schemas structure to perform the schema matching while some others approaches consider schema instances and some data mining technical to analyze dependences in the instances. We can consider these different methodologies and propose a generic method to produce semantic correspondences.

2.2. Mapping Generation and Adaptation in Peer-to-Peer Systems

Mapping generation consists in, given one target schema, finding a way to defining it from the available sources. Mapping adaptation adapts a given mapping for the target from the available sources. These methods can be used in many contexts such as data translation systems, wrappers, mediation systems, and data-warehouses. These systems follow the classical client/server architectures, in which there is a global view of all the available resources represented by the sources in the systems.

Different from the client/server architecture, the peer-to-peer architecture contains a set of peers have equivalent capabilities and responsibilities. There is no peer dedicated to serving the others and no peer has a global view of all the peers in the system. Every peer knows some neighbors. If it requests some information from the other peers, it first asks his neighbors, either one of these neighbors answer the request or they forward the request to their own neighbors and so on till one peer of the system answer the request.

If our mapping generation and adaptation approaches can be directly used in the client/server architecture for the target from the sources, it can also be used in a peer-to-peer system. Consider that one peer's request is represented by a target schema and mappings need to be generated for it from the other peers. To generate a mapping for this peer as well as to adapt an existing mapping for it, the process has to first try to produce mappings from its neighbors' schemas. If no mapping can be produced, then it considers other peers to generate mappings. However this leads to re-computation of the mapping generation and makes the whole process inefficient.

Our perspective is therefore to adapt our mapping generation and adaptation approaches to the peer-to-peer systems. The methods should be able to incrementally build the mappings for the target. For the mapping or mappings components that need to be found, some partial results may be first found from the neighbors of the target and then they will completed with the other peers. A cost model may be introduced and some optimization algorithm may be used to limit the full exploration of the system.

2.3. Schema and Correspondence Change Detection

In our approach of mapping adaptation, we consider a set of changes and we define a specific mapping adaptation process for every of the considered change. Few works has been proposed to detect change events. We have found only one approach [MAL05] that has been proposed to check the validation of semantic correspondences. It defines a set of computationally modules called sensors, which capture salient characteristics of data sources. The sensors are first trained and then deployed to detect broken correspondences.

Our perspective consists in developing a tool to detect the change events from the sources as well as from the target schema. It may analyze source instances for capture changes in the metadata. It may also analyze the system logs for the detection of the changes in the target schema. If a new version of a schema is given; the tool should also be able to analyse the difference between the two versions and return the set of change events representing the changes..

2.4. Global Change Propagation

The mapping adaptation approach presented in the thesis adapts the mapping for every change that the system considers. If several changes occur, they form a set of changes that the system will consider one by one. If many changes occur in the system, the change list may be long and the approach may become costly. Moreover, suppose that two changes occurred in the system: one removes a source element and the other adds the same element in another source. If the system first considers the removal, it may decide that the mapping is inadaptable, while the mapping may be still adapted if the system first considers the addition.

Our perspective consists in defining a global propagation process to propagate the schema changes to the mapping. Given a set of changes, partially ordered, the system has to consider globally all the changes and decides to applying the adaptation in an optimized order. For these purposes, a priority order may be defined for different change types. As an example, addition changes have intuitively more priority than remove changes. The system should also be able to analyze a given set of changes to re-order the changes having the same priority.

2.5. Quality Factors

Our mapping generation approach enumerates all the possible solutions to generate mappings and produces a set of mappings that represent different ways to define the target from the sources. As well as our mapping adaptation also consider enumerating all the possible adapted mappings. In case of a large number of source schemas, such method presents two drawbacks: (i) the time needed for enumerating all the possible combinations between the sources may be very large; (ii) many mappings may be generated which makes the choice of the "best" ones difficult.

Our perspective consists in taking into account some quality criteria to evaluate in the mapping generation or mapping adaptation. We consider three use cases. First, quality criteria can be used to classify the result mappings from either the mapping generation process or the mapping adaptation process. Quality factors are applied to analyze the different mappings with respect to the system expectation and allow the choice of the best one. For example, if the answering performance is the most important criteria for the system. Depending on the performances of the sources used in every mapping, an evaluation algorithm can be used to compute the answering performances of every mapping and therefore allow the classification of the mappings based on their performance. We have proposed a platform as a first attempt for evaluating data quality of mappings generated using our approach [KPS05].

In the second scenario, we consider every user may have some user preferences concerning the quality of the query results. The system conserves all the result mappings in the system. Every time a user sends a query to the target schema, the system analyzes the system's mappings with respect to this user's preference and selects one mapping that satisfies the most the user preferences to rewrite the user query. As an example, if a user expresses his preference of using a particular source, then mappings involving this source will have more priority than the others. This capability of adapting system behaviours for different users is also called system polymorphism [BK06].

The third scenario is quality driven mapping generation and adaptation. It consists in using some quality evaluation inside the mapping generation and the mapping adaptation processes. The evaluation process can be executed after every step of the generation or adaptation to eliminate some intermediate results that will give mappings of unsatisfying qualities. This will limit the search spaces of the generation and adaptation then optimize the performance of the systems.

For all these possible scenarios, we need to define a set of quality criteria considered by the system. Two algorithms of propagation need to be specified: one propagates the source qualities to mappings and the other propagates the user expectations to mappings. We also need to define the evaluation process to be able to evaluate mappings based on their qualities. One approach [PRB04] has been proposed for the quality evaluation based on database freshness and accuracy. There is also an on-going work on user preferences specification and managements [Kos06]. We need to define the algorithms to evaluate mappings with respect to user or system expectation as well as the way to consider it in the process of mapping generation and adaptation.

Appendix I. XML Schema Representation of the Three Schemas in Figure 3-1

In this section, we show the representations in XML Schema of the three schemas considered in Figure 3-1 of Chapter 3:

The Target Schema TS

```
<schema id="TS" xmlns="http://www.w3.org/2001/XMLSchema"
      elementFormDefault="qualified" attributeFormDefault="unqualified">
  <element name="library">
    <complexType>
      <sequence>
        <element name="author" minOccurs="1" maxOccurs="unbounded">
          <complexType>
              <sequence>
                <element name="id" type="xs:string"/>
                <element name="name" type="xs:string"/>
                <element name="address" type="xs:string"/>
                <element name="book" minOccurs="1" maxOccurs="unbounded">
                  <complexType>
                    <sequence>
                      <element name="isbn" type="xs:string"/>
                      <element name="booktitle" type="xs:string"/>
                      <element name="priceineuro" type="xs:string"/>
                      <element name="Chapter" minOccurs="1" maxOccurs="unbounded">
                        <complexType>
                          <sequence>
                            <element name="number" type="xs:string"/>
                            <element name="chaptertitle" type="xs:string"/>
                            <element name="abstract" minOccurs="0" type="xs:string"/>
                          </sequence>
                        </complexType>
                      </element>
                    </sequence>
                  </complexType>
                  <key name="chapterkey">
                    <selector xpath="chapter"/>
                    <field xpath="number"/>
                </key>
              </element>
            </sequence>
          </complexType>
        </element>
      </sequence>
    </complexType>
    <key name="bookKey">
      <selector xpath="author/book"/>
      <field xpath="isbn"/>
    </key>
    <key name="authorKey">
      <selector xpath="author"/>
      <field xpath="id"/>
    </key>
   </element>
</schema>
```

The Source Schema S1

```
<schema id="S1" xmlns="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified"
attributeFormDefault="unqualified">
  <element name="library">
    <complexType>
      <sequence>
        <element name="author" minOccurs="1" maxOccurs="unbounded">
          <complexType>
            <sequence>
              <element name="firstname" type="xs:string"/>
              <element name="lastname" type="xs:string"/>
              <element name="id" type="xs:string"/>
            </sequence>
          </complexType>
        </element>
            <element name="book" minOccurs="1" maxOccurs="unbounded">
          <complexType>
            <sequence>
              <element name="isbn" type="xs:string"/>
              <element name="booktitle" type="xs:string"/>
              <element name="chapter" minOccurs="1" maxOccurs="unbounded">
                <complexType>
                  <sequence>
                    <element name="number" type="xs:string"/>
                    <element name="abstract" type="xs:string"/>
                  </sequence>
                </complexType>
              </element>
              </sequence>
            </complexType>
          </element>
          <element name="address" minOccurs="1" maxOccurs="unbounded">
          <complexType>
            <sequence>
              <element name="authorid" type="xs:string"/>
              <element name="authoraddress" type="xs:string"/>
            </sequence>
          </complexType>
          </element>
        </sequence>
    </complexType>
    <key name="bookkey">
      <selector xpath="author/book"/>
      <field xpath="isbn"/>
    </key>
    <key name="authorkey">
      <selector xpath="author"/>
      <field xpath="id"/>
    </key>
    <keyref name="addresskeyref" refer="authorkey" >
      <selector xpath="address"/>
      <field xpath="authorid"/>
    </keyref>
   </element>
</schema>
```

The Source Schema S2

```
<schema id="S2" xmlns="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified"
attributeFormDefault="unqualified">
  <element name="library">
    <complexType>
      <sequence>
        <element name="book" minOccurs="1" maxOccurs="unbounded">
          <complexType>
            <sequence>
              <element name="isbn" type="xs:string"/>
              <element name="booktitle" type="xs:string"/>
              <element name="priceindollar" type="xs:string"/>
              <element name="authorid" type="xs:string" maxOccurs="unbounded"/>
              <element name="chapter" minOccurs="1" maxOccurs="unbounded">
                <complexType>
                  <sequence>
                    <element name="number" type="xs:string"/>
                    <element name="chaptertitle" type="xs:string"/>
                  </sequence>
                </complexType>
                </element>
            </sequence>
          </complexType>
          <key name="chapterkey">
            <selector xpath="chapter"/>
            <field xpath="number"/>
          </key>
          </element>
        </sequence>
    </complexType>
    <key name="bookkey">
      <selector xpath="book"/>
      <field xpath="isbn"/>
    </key>
  </element>
</schema>
```

Appendix II. Transformation of some Typical Mappings into our Representation

W3C published a document of XML query use cases [xquc]. The use case "XMP" contains 12 sample queries to represent from most easy to more complicated use cases. We show the transformation of the queries 1-5, 7, 9 into our XQuery representation. The other 5 queries contain more complicated expression such as `if-else` clauses and restrictions on tag name etc. This will be one of our perspectives to extend our representation to be able to support these XQuery features.

In our approach of mapping adaptation, we consider only elements for simplicity. However, the same process can be applied in considering both elements and attributes: attributes can be considered as monovalued elements that can be optional or mandatory. In schema and mapping representations, there is only syntactical difference between elements and attributes. The sample queries we will use as follows contain both attributes and elements. We can see that our transformation applies to the sample queries in a way that only syntactical difference exists between the attributes and the elements.

Query 1

The query lists books published by Addison-Wesley after 1991, including their year and title.

```
<bib>{
  for $b in doc("http://bstore1.example.com/bib.xml")/bib/book
  where $b/publisher = "Addison-Wesley" and $b/@year > 1991
  return
    <book year="{ $b/@year }">
     { $b/title }
    </book>
 }</bib>
```

This query can be transformed into our internal representation as follows. We do not consider selections (e.g. $b/publisher = "Addison-Wesley") in our query representation for simplicity. But the extension of our approach to support selections is straightforward.

```
<bib>{
  for $b in doc("http://bstore1.example.com/bib.xml")/bib/book
  where $b/publisher = "Addison-Wesley" and $b/@year > 1991
  return
   <book year="{ data($b/@year)}">{
        <title>{data($b/title)}</title>
   }</book>
 }</bib>
```

Query 2

This query creates a flat list of all the title-author pairs, with each pair enclosed in a "result" element.

```
<results> {
    for $b in doc("http://bstore1.example.com/bib.xml")/bib/book,
        $t in $b/title,
        $a in $b/author
```

```
   return
      <result>
         { $t }
         { $a }
      </result>
 }</results>
```

The query can be transformed into our internal representation as follows:

```
<results> {
   for $b in doc("http://bstore1.example.com/bib.xml")/bib/book/author
   return
      <result>
         <title>{ data($t/PARENT::book/title)}</title>
         <author>{ data($a) }</author>
      </result>
 }</results>
```

Query 3

This query lists, for each book in the bibliography, its title and authors, grouped inside a "result" element.

```
<results>{
   for $b in doc("http://bstore1.example.com/bib.xml")/bib/book
   return
      <result>
         { $b/title }
         { $b/author  }
      </result>
}</results>
```

The query can be transformed into our internal representation as follows:

```
<results> {
   for $b in doc("http://bstore1.example.com/bib.xml")/bib/book
   return
      <result>
         <title>{ data($b/title)}</title>
        for $a in doc("http://bstore1.example.com/bib.xml")/bib/book/author
        where $b = $a/PARENT::book
         return <author>{ data($a) }</author>
      </result>
 }</results>
```

Query 4

This query lists, for each author in the bibliography, the author's name and the titles of all books by that author, grouped inside a "result" element.

We do not consider ordering in our approach. This can be considered as a post-process to reorder the result of the rest of the query. Here we keep the ordering function in the transformation but it will not be considered in the mapping adaptation.

```
<results>{
   let $a := doc("http://bstore1.example.com/bib/bib.xml")//author
   for $last in distinct-values($a/last),
      $first in distinct-values($a[last=$last]/first)
   order by $last, $first
   return
```

```
        <result>
           <author>
             <last>{ $last }</last>
             <first>{ $first }</first>
           </author>{
              for $b in doc("http://bstore1.example.com/bib.xml")/bib/book
              where some $ba in $b/author
                   satisfies ($ba/last = $last and $ba/first=$first)
              return $b/title
           }</result>
  }</results>
```

The query can be transformed into our internal representation as follows:

```
<results>{
   for $last in distinct-values doc("http://bstore1.example.com/bib/bib.xml")//author/last),
   for $first in distinct-values(("http://bstore1.example.com/bib/bib.xml")//author/[last=$last]/first)
   for $author in "http://bstore1.example.com/bib/bib.xml")//author
   where $author/last = $last and $author/first = $first
   order by $last, $first
   return
      <result>{
         <author>{
           <last>{ data($author/last) }</last>
           <first>{ data($author/first) }</first>
         }</author>
         for $b in doc("http://bstore1.example.com/bib.xml")/bib/book
         where $b/author[first=$author/first]/last = $author/last
            return $b/title
         }</result>
  }</results>
```

Query 5

This query for each book found at both bstore1.example.com and bstore2.example.com, list the title of the book and its price from each source.

```
<books-with-prices> {
   for $b in doc("http://bstore1.example.com/bib.xml")//book,
      $a in doc("http://bstore2.example.com/reviews.xml")//entry
   where $b/title = $a/title
   return
      <book-with-prices>
         { $b/title }
         <price-bstore2>{ $a/price/text() }</price-bstore2>
         <price-bstore1>{ $b/price/text() }</price-bstore1>
      </book-with-prices>
  }</books-with-prices>
```

The query can be transformed into our internal representation as follows:

```
<books-with-prices> {
   for $b in doc("http://bstore1.example.com/bib.xml")//book,
      $a in doc("http://bstore2.example.com/reviews.xml")//entry
   where $b/title = $a/title
   return
      <book-with-prices>
         <title>{ data($b/title) }</title>
         <price-bstore2>{ data($a/price) }</price-bstore2>
```

```
          <price-bstore1>{ data($b/price) }</price-bstore1>
        </book-with-prices>
  }</books-with-prices>
```

Query 7

This query lists the titles and years of all books published by Addison-Wesley after 1991, in alphabetic order.

```
<bib>{
    for $b in doc("http://bstore1.example.com/bib.xml")//book
    where $b/publisher = "Addison-Wesley" and $b/@year > 1991
    order by $b/title
    return
        <book>
            { $b/@year }
            { $b/title }
        </book>
  }</bib>
```

The query is transformed into our representation as follows. As query 4, we keep the ordering function in the transformation but it will not be considered in the mapping adaptation.

```
<bib>{
    for $b in doc("http://bstore1.example.com/bib.xml")//book
    where $b/publisher = "Addison-Wesley" and $b/@year > 1991
    order by $b/title
    return
        <book>
            <year>{ data($b/@year) }</year>
            <title>{ data($b/title) }</title>
        </book>
  }</bib>
```

Query 9

This query finds all section or chapter titles that contain the word "XML", regardless of the level of nesting.

```
<results>{
    for $t in doc("books.xml")//(chapter | section)/title
    where contains($t/text(), "XML")
    return $t
  }</results>
```

The query is transformed into our representation as follows.

```
<results>{
    for $t in doc("books.xml")//(chapter | section)/title
    where contains($t/text(), "XML")
   return <title>{data($t)}</title>
}</results>
```

References

[ACB06] Paolo Atzeni, Paolo Cappellari, Philip A. Bernstein: Model-Independent Schema and Data Translation. EDBT 2006: 368-385

[ACM97] Serge Abiteboul, Sophie Cluet, Tova Milo: Correspondence and Translation for Heterogeneous Data. ICDT 1997: 351-363

[AIS] Adeptia Integration Server (AIS): http://www.adeptia.com/products/ais.html

[AMF] Altova MapForce: http://www.altova.com/products_mapforce.html

[ABM05a] Yuan An, Alexander Borgida, John Mylopoulos: Inferring Complex Semantic Mappings Between Relational Tables and Ontologies from Simple Correspondences. ODBase'05: 1152-1169

[ABM05b] Yuan An, Alexander Borgida, John Mylopoulos: Constructing Complex Semantic Mappings Between XML Data and Ontologies. International Semantic Web Conference 2005: 6-20

[AMR98] Serge Abiteboul, Jason McHugh, Michael Rys, Vasilis Vassalos, Janet L. Wiener: Incremental Maintenance for Materialized Views over Semistructured Data. Proc. of the 24th Int. Conf. on Very Large Data Bases (VLDB'98), New York, USA (1998) 38-49

[Bil79] Horst Biller: On the equivalence of data base schemas - a semantic approach to data translation. Inf. Syst. 4(3): 35-47 (1979)

[BH] Blue Phoenix Datamigrator: http://www.bphx.com/data_migration_tool.cfm?source=gada_datamig

[BKL04] Michael Boyd, Sasivimol Kittivoravitkul, Charalambos Lazanitis, Peter McBrien, Nikos Rizopoulos: AutoMed: A BAV Data Integration System for Heterogeneous Data Sources, Proc. of CAiSE04, 2004: 82-97

[BLK03] Mokrane Bouzeghoub, Bernadette Farias Lóscio, Zoubida Kedad, Ana Carolina Salgado: Managing the Evolution of Mediation Queries. Proc. of the 11th Int. Conf. on Cooperative Information Systems (CoopIS'03), Catania, Sicily, Italy (2003) 22-37

[BLN86] Carlo Batini, Maurizio Lenzerini, Shamkant B. Navathe: A Comparative Analysis of Methodologies for Database Schema Integration. ACM Comput. Surv. 18(4): 323-364 (1986)

[BKS04] Mokrane Bouzeghoub, Zoubida Kedad, Assia Soukane: Improving Mediation Query Generation Using Constraints and Metadata, in Proc. of BDA 2004: 385-406

[BLN86] Carlo Batini, Maurizio Lenzerini, Shamkant B. Navathe: A Comparative Analysis of Methodologies for Database Schema Integration. ACM Comput. Surv. 18(4): 323-364 (1986)

[CG04] Paulo J. F. Carreira, Helena Galhardas: Efficient Development of Data Migration Transformations. SIGMOD Conference 2004: 915-916

[CLL03] Ya Bing Chen, Tok Wang Ling, Mong-Li Lee: Automatic Generation of XQuery View Definitions from ORA-SS Views. ER 2003: 158-171

[CR03] Kajal T. Claypool, Elke A. Rundensteiner: Sangam: A Transformation Modeling Framework. In Proc. of DASFAA 2003: 47-54

[CBB03a] Christine Collet, Khalid Belhajjame, Gilles Bernot, Gennaro Bruno, Christophe Bobineau, Béatrice Finance, Fabrice Jouanot, Zoubida Kedad, David Laurent, Genoveva Vergas-Solar, Tuyet-Trinh Vu, Xiaohui Xue: Mediagrid : a mediation framework for a transparent access to biological data sources, Poster in the 22nd International Conference on Conceptual Modeling (ER'03), Chicago, Illinois, October 2003

[CBB03b] Christine Collet, Khalid Belhajjame, Gilles Bernot, Gennaro Bruno, Christophe Bobineau, Beatrice Finance, Fabrice Jouanot, Zoubida Kedad, David Laurent, Genoveva Vergas-Solar, Tuyet-Trinh Vu, Xiaohui Xue: Towards a target system framework for transparent access to largely distributed sources, in the European Conf. on Computational Biology (ECCB'03), Paris, France, June 2004

[CBB04] Christine Collet, Khalid Belhajjame, Gilles Bernot, Gennaro Bruno, Christophe Bobineau, Béatrice Finance, Fabrice Jouanot, Zoubida Kedad, David Laurent, Genoveva Vergas-Solar, Tuyet-Trinh Vu, Xiaohui Xue: Towards a target system framework for transparent access to largely distributed sources, in the Int. Conf. on Semantics of a Networked World Semantics for Grid Databases (IC-SNW'04), Paris, France, June 2004, 65-78

[DLD04] Dhamankar, R., Lee, Y., Doan, A., Halevy, A. Y., Domingos, P.: iMAP: Discovering Complex Mappings between Database Schemas. Proc. of Int. Conf. ACM SIGMOD (SIGMOD'04), Paris, France (2004) 383-394

[dom] W3C, Document Object Model (DOM), http://www.w3.org/TR/DOM-Level-3-Core/

[DR02] Do, H.H., Rahm, E.: COMA - A System for Flexible Combination of Schema Matching Approaches. Proc. VLDB, Hong Kong, China (2002) 610-621

[dtd] W3C, Extensible Markup Language (XML) 1.0 (Fourth Edition), Section 2.8 Prolog and Document Type Declaration, http://www.w3.org/TR/xml/#sec-prolog-dtd

[Gal06] Gal, A.: Managing Uncertainty in Schema Matching with Top-K Schema Mappings. Proc of Journal on Data Semantics (2006)

[DWL00] Gillian Dobbie, Wu Xiaoying, Tok Wang Ling, Mong Li Lee: ORA-SS: An Object-Relationship-Attribute Model for Semistructured Data TR21/00, Technical Report, Department of Computer Science, National University of Singapore, December 2000.

[FP04] Hao Fan, Alexandra Poulovassilis: Schema Evolution in Data Warehousing Environments - A Schema Transformation-Based Approach. ER 2004: 639-653

[FW06] George H. L. Fletcher, Catharine M. Wyss: Data Mapping as Search. EDBT 2006: 95-111

[FS03] Bernadette Farias Lóscio, Ana Carolina Salgado: Generating Mediation Queries for XML-based Data Integration Systems. SBBD 2003: 99-113

[FS04] Bernadette Farias Lóscio, Ana Carolina Salgado: Evolution of XML-Based Mediation Queries in a Data Integration System. Proc. of ER Workshops, Shanghai, China (2004): 402-414

[FSG03] Bernadette Farias Lóscio, Ana Carolina Salgado, Luciano do Rêgo Galvão: Conceptual modeling of XML schemas. WIDM 2003: 102-105

[GMR95] Ashish Gupta, Inderpal Singh Mumick, Kenneth A. Ross: Adapting Materialized Views after Redefinitions. Proc. of Int. Conf. of ACM SIGMOD (SIGMOD'95), San Jose, California (1995): 211-222

[GPQ97] H. Garcia-Molina , Y. Papakonstantinou , D. Quass , A. Rajaraman , Y. Sagiv , J. Ullman , V. Vassalos , J. Widom: The TSIMMIS approach to mediation: Data models and Languages. In Journal of Intelligent Information Systems, 1997.

[HC06] Bin He and Kevin Chen-Chuan Chang: Automatic Complex Schema Matching across Web Query Interfaces: A Correlation Mining Approach. Proc. of ACM Transactions on Database Systems (2006)

[Inm96] Inmon, W. Building the Data Warehouse. Wiley & Sons, New York, 1996.

[IS06] Utku Irmak, Torsten Suel: Interactive wrapper generation with minimal user effort. WWW 2006: 553-563

[jdbc] Java Database Connectivity (JDBC): http://java.sun.com/javase/technologies/database.jsp

[KA04] Anastasios Kementsietsidis, Marcelo Arenas: Data Sharing Through Query Translation in Autonomous Sources. VLDB 2004: 468-479

[KB99] Zoubida Kedad, Mokrane Bouzeghoub: Discovering View Expressions from a Multi-Source Information System. in Proc. of the 7th Int. Conf. on Cooperative Information Systems (CoopIS'99) 57-68

[KR02] Andreas Koeller, Elke A. Rundensteiner: Incremental Maintenance of Schema-Restructuring Views. in Proc. of 8th Int. Conf. of Extending Database Technology (EDBT'02), Prague, Czech Republic, (2002) 354-371

[KX05a] Zoubida Kedad, Xiaohui Xue: An automatic tool for discovering complex mappings, Technical Report #2005/75, University of Versailles, Versailles, France, 2005

[KX05b] Zoubida Kedad, Xiaohui Xue: Mapping Generation for XML Data Sources: a General Framework, in the Int. Workshop on Challenges in Web Information Retrieval and Integration (WIRI'05), in conjunction with the 21st Int. Conf. on Data Engineering (ICDE'05), Tokyo, Japan, April (2005) 164-172

[KX05c] Zoubida Kedad, Xiaohui Xue: Discoverying complex mappings for XML data integration, in the 21th Conf. of Bases de données avancées (BDA'05), Saint Malo, France, October (2005) 41-50

[KX05d] Zoubida Kedad, Xiaohui Xue: Mapping Discovery for XML Data Integration. in Proc. of the 13th Int. Conf. on Cooperative Information Systems (CoopIS'05), Agia Napa, Cyprus, November (2005) 166-182

[Kos06] Dimitre Kostadinov: Personalization de l'information et gestion de profils utilisateur, PhD Thesis, University of Versailles Saint-Quentin-en-Yvelines (Ongoing work)

[KPS04] Dimitre Kostadinov, Verónika Peralta, Assia Soukane, Xiaohui Xue: Système adaptif à l'aide de la génération de requêtes de médiation (French), Demonstration in the 20th Conf. of Bases de données avancées (BDA'04), Montpellier, France, Ocotober 2004, 351-355

[KPS05] Kostadinov, D., Peralta, V., Soukane, A., Xue, X.: Intégration de données hétérogènes basée sur la qualité. Proc. of INFORSID 2005 (Inforsid'05), Grenoble, France (2005) 471-486

[Len02] Lenzerini, M.: Data integration: a theoretical perspective. In Proc. of PODS, (2002) 233–246

[LD94] Qing Li, Guozhu Dong: A Framework for Object Migration in Object-Oriented Databases. Data Knowl. Eng. 13(3): 221-242 (1994)

[LNR02] Amy J. Lee, Anisoara Nica, Elke A. Rundensteiner: The EVE Approach: View Synchronization in Dynamic Distributed Environments. IEEE Trans. Knowl. Data Eng. 14(5): 931-954 (2002)

[LPH00] Ling Liu, Calton Pu, Wei Han. `` XWRAP: An XML-enabled Wrapper Construction System for Web Information Sources", Proceedings of the 16th International Conference on Data Engineering (ICDE'2000), San Diego CA (February 28 - March 3, 2000)

[MAL05] Robert McCann, Bedoor K. AlShebli, Quoc Le, Hoa Nguyen, Long Vu, AnHai Doan: Mapping Maintenance for Data Integration Systems. VLDB 2005: 1018-1030

[MHH00] Renée J. Miller, Laura M. Haas, Mauricio A. Hernández: Schema Mapping as Query Discovery. Proc. of the 26th Int. Conf. on Very Large Data Bases (VLDB'00), Cairo, Egypt (2000) 77-88

[MP02] Peter McBrien, Alexandra Poulovassilis: Schema Evolution in Heterogeneous Database Architectures, A Schema Transformation Approach. CAiSE 2002: 484-499

[MRB03] Sergey Melnik, Erhard Rahm, Philip A. Bernstein: Rondo: A Programming Platform for Generic Model Management. SIGMOD Conference 2003: 193-204

[NR99] Anisoara Nica, Elke A. Rundensteiner: View Maintenance after View Synchronization. Proc. of Int. Database Engineering and Applications Symposium (IDEAS'99), Montreal, Canada (1999) 215-213

[Per06] Verónika Peralta: Data Quality Evaluation in Data Integration Systems, PhD Thesis, University of Versailles Saint-Quentin-en-Yvelines (Ongoing work)

[PRB04] Verónika Peralta, Raul Ruggia, Mokrane Bouzeghoub: Analyzing and Evaluating Data Freshness in Data Integration Systems. Journal of Ingénierie des Systèmes d'Information 9(5-6): 145-162 (2004)

[PT99] Lucian Popa, Val Tannen: An Equational Chase for Path-Conjunctive Queries, Constraints, and Views. Proc. of the 7th Int. Conf. on Database Theory (ICDT'99), Jerusalem, Israel (1999): 39-57

[PVM02] Lucian Popa, Yannis Velegrakis, Renée J. Miller, Hernandez M.A., Fagin R.: Translating web data. Proc. of the 28th Int. Conf. on Very Large Data Bases (VLDB'02), Hong Kong, China (2002) 598-609

[RB01] E. Rahm, P. A. Bernstein, A survey of approaches to automatic schema matching. Proc. of the 27th Int. Conf. on Very Large Data Bases (VLDB'01), Roma, Italy (2001) 334-350

[RSV01] Chantal Reynaud, Jean-Pierre Sirot, Dan Vodislav: Semantic Integration of XML Heterogeneous Data Sources. IDEAS 2001: 199-208

[SE05] Shvaiko, P., Euzenat, J.: A Survey of Schema-based Matching Approaches . Proc. of Journal on Data Semantics IV (2005) 146-171

[SKS01] Jayavel Shanmugasundaram, Jerry Kiernan, Eugene J. Shekita, Catalina Fan, John E. Funderburk: Querying XML Views of Relational Data. VLDB 2001: 261-270

[Sou05] Assia Soukane: Génération automatique des requêtes de médiation avec prise en compte des besoins des utilisateurs dans un environnement hétérogène. PhD Thesis, Université de Versailles-Saint-Quentin-en-Yvelines, December 2005

[Sty] Stylus XML-to-XML Mapper: http://www.stylusstudio.com/xml_to_xml_mapper.html/

[SKR01] Hong Su, Harumi A. Kuno, Elke A. Rundensteiner: Automating the transformation of XML documents. WIDM 2001: 68-75

[SF] SourceForge.net: http://sourceforge.net/

[THH05] Philippe Thiran, Jean-Luc Hainaut, Geert-Jan Houben: Database Wrappers Development: Towards Automatic Generation. Proc. of the 9th European Conference on Software Maintenance and Reengineering (CSMR'05), Manchester, UK (2005): 207-216

[TRV98] Anthony Tomasic, Louiqa Raschid, Patrick Valduriez: Scaling Access to Heterogeneous Data Sources with DISCO. IEEE Trans. Knowl. Data Eng. 10(5): 808-823 (1998)

[TS99] Dimitri Theodoratos, Timos K. Sellis: Designing Data Warehouses. Data Knowl. Eng. 31(3): 279-301 (1999)

[VMP04] Yannis Velegrakis, Renée J. Miller, Lucian Popa: Preserving mapping consistency under schema changes. VLDB J. 13(3): 274-293 (2004)

[Wie92] Wiederhold, G.: "Mediators in the architecture of future information systems". IEEE Computer, Vol. 25(3):38-49, 1992.

[WN] WordNet: a Lexical Database for English: http://wordnet.princeton.edu/

[WR05] Catharine M. Wyss, Edward L. Robertson: Relational languages for metadata integration. ACM Trans. Database Syst. 30(2): 624-660 (2005)

[xerces] Xerces2 Java Parser: http://xerces.apache.org/xerces2-j/

[xml] W3C, eXtensible Markup Language (XML), http://www.w3.org/XML/

[xpath] W3C, XML Path Language 1.0, http://www.w3.org/TR/xpath/

[xquery] W3C, XQuery 1.0, an XML Query Language, http://www.w3.org/TR/xquery/

[xquc] W3C, XML Query Use Cases, http://www.w3.org/TR/xquery-use-cases/

[xsd] W3C, XML Schema, http://www.w3.org/XML/Schema/

[xslt] W3C, XSL Transformations (XSLT), http://www.w3.org/TR/xslt/

[Xue06] Xiaohui Xue: Adapting Mapping for Schema Evolution: An Automatic Approach. Internal Technical Report, Sial Team, University of Versailles, April, 2006

[YLL03] Xia Yang, Mong-Li Lee, Tok Wang Ling: Resolving Structural Conflicts in the Integration of XML Schemas: A Semantic Approach. ER 2003: 520-533

[YP04] Cong Yu and Lucian Popa. Constraint-Based XML Query Rewriting for Data Integration. Proc. of Int. Conf. of ACM SIGMOD, Paris, France (2004): 371-382

[YP05] Cong Yu, Lucian Popa: Semantic Adaptation of Schema Mapping when Schemas Evolve. Proc. of the 31st Int. Conf. on Very Large Data Bases (VLDB'05), Trondheim, Norway, (2005)

[Zam04] Lucas Zamboulis: XML Data Integration by Graph Restructuring. BNCOD (2004) 57-71