Transformation-based Approach to Resolving Data Heterogeneity

by

Yury Alexandrovich Bychkov
B.Sc., University of Victoria, 1999

A Thesis Submitted in Partial Fulfillment of the Requirements
for the Degree of

MASTER OF SCIENCES

in the Department of Computer Science

© Yury Alexandrovich Bychkov, 2004
University of Victoria

All rights reserved. This thesis may not be reproduced in whole or in part by photocopy or other means, without the permission of the author.

Supervisor: Dr. Jens Jahnke

ABSTRACT

The amount of electronic medical data that has appeared in the last decade is enormous. However, there exists a big gap between the potential and the realized value of such data, mainly because it is contained within isolated Medical Information Systems (MISs). Thus, enabling data interchange between MISs would improve the quality of healthcare services and even allow medical organizations to offer new services that were impossible or impractical before. Unfortunately, these information systems are highly heterogeneous and, in order for the data to be exchanged, this *representational heterogeneity* has to be resolved.

The main objective of this thesis was to develop an approach to specifying the translation between the aforementioned heterogenous data sources. This translation specification is comprised from the sequence of transformations that have been formally defined and the effect of which on the information capacity of the schemas is fully known.

Table of Contents

List of Figures

List of Tables

Acknowledgement

I would like to express my sincere gratitude to my supervisor, Dr. Jens Jahnke for his support and guidance during my studies and patience with my tendency to be distracted with other projects. Thanks to my friends and colleagues from the NetLab group (especially Christina Obry, David Dahlem, Adeniyi Onabajo, Glen McCallum, Iryna Bilykh and Barbara Kursawe) for the productive and stimulating discussions (even if they were not always work-related) that helped shape this research and from the School of Health Information Sciences (Dr. Francis Lau, Craig Kuziemsky and Rebecca Westle) for sharing their domain expertise. Special thanks to my family and friends for all their support and encouragement over the years.

Dedication

To my parents

Chapter 1

Introduction

1.1 Motivation

Over the last decades, the drastic reduction in computing costs coupled with an increased availability of affordable broadband connectivity have created an ocean of electronic data spread over a large number of information systems. However, there is a considerable gap between the potential value of this data and the reality. While many such systems are accessible via the Internet standards, this access is usually restricted to entering or viewing the data using a human-targeted user interface, which limits the usefulness of these information systems.

Solving this problem by enabling the information systems to connect to each other is especially crucial in areas like healthcare, since the more complete the information available (even simply by showing a patient's data across hospital systems, instead of only seeing one system at a time (e.g., pharmacy or charting)) is to the doctors, the more lives could be saved or improved. Because of this, in recent years, a considerable effort by both governmental and private organizations around the world (e.g., Canada Health Infoway [1], established by the Canadian government in 2000) has been devoted

to improving healthcare services by integrating medical information systems (MISs) in order to provide an Electronic Medical Record (EMR) infrastructure.

Unfortunately, while desirable, integrating MISs is not an easy task. As Walter Sujansky says in [60]:

> *In aggregate, these data encompass information and knowledge that can significantly improve patient care, public health, basic research, and administrative efficiency. However, the wonderful volume and availability of these data have grown through a largely decentralized process ...[that] has resulted in a patchwork of diverse, or heterogeneous, database implementations, making access to and aggregation of data across implementations very difficult from a practical perspective.*

1.2 The HealthMatrix project

The *HealthMatrix* project[1](described in details in Chapter 3) attempts to solve the integration problem by acting as a middleware that can facilitate data and service interchange between existing medical information systems. With the help of *HealthMatrix*, MISs can be connected together to form large scale distributed networks (that can be reconfigured on-the-fly). To achieve this, *HealthMatrix* uses active components to:

- wrap the access to the heterogenous MISs and provide a uniform query/data

[1]developed by the NetLab Group, Department of Computer Science, University of Victoria

interface [SFE, cf. Section 3.3.1]

- route the data/queries over the network, depending on the purpose of said data [Staging Area, cf. Section 3.3.2.2]
- manipulate the transmitted data [Transducer/Merger/Adder, cf. Section 3.3.2.1, 3.3.2.5]
- execute predefined guidelines (at the MISs) in order to obtain additional data [WEE, cf. Section 3.3.2.6]
- provide a central repository and control center for each distributed network [MEA, cf. Section 3.3.3]

1.3 Data Heterogeneity

Unfortunately, it is not enough to connect the information systems together to enable the information exchange between then. One of the problems addressed by *Health-Matrix*, and the focus of this thesis, is that of resolving data heterogeneity. The main reason for the information systems containing similar information to be heterogeneous is the fact they *"have grown through a largely decentralized process that has allowed organizations to meet specific or local data needs without requiring them to coordinate and standardize their database implementations"* [60]. This data heterogeneity can take many forms [cf. Section 2.2] that range from different name used for the same concept to what the data means and how it is structured.

In order to resolve the heterogeneity between two data sources and translate data

from one representation to another, it is necessary to: (a) figure out which element(s) in one source match which element(s) in another and (b) create a specification (or program) that contains instructions on how to translate the data. In this thesis we are addressing the latter problem.

Unlike the majority of existing approaches [cf. Section 4.2.2] to the translation specification that are based on mapping the elements from the source representation to elements from the target representation (in some cases using various functions), our approach (explained in detail in Chapters 4 and 5) relies on transforming the source into the target by applying a sequence of formally defined transformations with well-known properties to it. It is our contention that, while being at least as powerful as the majority of mapping approaches, our approach also allows reasoning about the information loss/gain during the translation (and this is especially important in a healthcare domain).

1.4 Outline

The rest of the thesis is structured as follows:

Chapter 2 describes background research for this work. It introduces our classification of data heterogeneity and describes markup languages and their history as well as the background on graph transformations.

Chapter 3 presents the *HealthMatrix* project. It discusses the requirements for the health information networks and describes individual components of the

HealthMatrix and interaction between them.

Chapter 4 outlines the problem of data translation. It describes the existing approaches to schema matching and translation specifications and presents our transformation-based approach to resolving data heterogeneity.

Chapter 5 defines the transformation operations and their properties. It also describes our method of schema representation as well as several approaches designed to make the creation of the translation specification easier for the user.

Chapter 6 outlines the prototype implementation and evaluates our approach with respect to the data heterogeneity classification (presented in Chapter 2) and the *palliative care data mapping* case study.

Chapter 7 presents the summary of the contributions of this work and talks about future research directions.

Chapter 2

Background

2.1 XML and Markup Languages

2.1.1 History

2.1.1.1 Early Days

History of the markup languages is quite long. The oldest of such languages predate computer science and were used to mark up hand-written manuscripts for printing. Different printing houses used their own sets of markup symbols until the first widely known common set was introduced in "Rules for Compositors and Readers, which are to be observed in all cases where no special instructions are given" by Horace Hart (informally called "Hart's Rules") [39] that was published in 1893 and continued to be very popular (39 editions, last one in 1983) up until the advent of computer-based publishing.

As printing became more computerized, a need for computer markup languages arose. Unfortunately, every word-processing program had its own markup language and most of these languages used *specific* (also called *procedural*) *markup.*

Specific (procedural) markup instructs the processing system how a document

should be presented on screen or in print (sometimes a separate markup is required for each target). It is concerned only with formatting and does not provide any information on the meaning of the marked-up sections. For example, the instructions in Figure 2.1 simply makes the text (which is a chapter's heading, but the language has no means of specifying that) to be centered, bold and in "roman16" font.

```
.sp
.fs roman16
.bd .ct Chapter 1. Introduction
```

Figure 2.1. *Example of the* specific markup

2.1.1.2 GenCode

In 1967, the president of the Composition Committee of the Graphic Communication Association (GCA) William Tunnicliffe presents for the first time the idea of separation between the contents and the formatting of a document. At approximately the same time, Stanley Rice (a book designer from New York) proposes to use a universal set of parameterizable tags to describe a so-called *editorial structure* of the document. These two ideas lead to the creation of the GenCode Committee, conclusions of which became the foundations of modern markup languages:

- It is impossible to describe all the documents with one set of codes
- Markup should take into account the hierarchical structure of the document
- Markup should be *generic* (or *descriptive*) rather than *specific*

Generic (descriptive) markup identifies the structure and components of the document. It is concerned with what a particular structural part of the document means, rather than with how it should be presented (thus it requires other processes to provide a specific formatting for each type of document's component). For example, the instructions in Figure 2.2 simply mark the text as the chapter's heading (and this, if we want to use this information for the formatting, might mean formatting it, for example, as bold and centered or italicized and left-aligned, depending on a settings of the processing application). Thus, the *generic markup* is much more powerful than the *specific markup*, since it is not targeted towards any particular purpose (such as formatting) and can be used in a variety of ways.

```
:chapter Chapter 1. Introduction
```

Figure 2.2. *Example of the* generic markup

2.1.1.3 GML

In 1969, as part of IBM's project on integrating law office information subsystems, Charles Goldfarb extended the ideas of GenCode committee and together with Edward Mosher and Ray Lorie created the Generalized Markup Language (GML) as a means of allowing the text editing, formatting, and information retrieval subsystems to share documents. Note that originally GML stood only for the initials of its creators, until Goldfarb coined the term *"markup language"* in [38]. GML started an entirely new layer in markup languages, since it was actually a *metalanguage* (i.e.,

a language for describing other languages). Also, unlike earlier markup languages, instead of a simple tagging system, GML introduced the concept of a formally-defined document type with an explicit nested element structure.

Recognizing in GML the value beyond law office applications, IBM retargeted it to text processing in general. Since GML was a metalanguage, in order for it to be used by IBM's Document Composition Facility, a set of tags (called GML Starter Set [9]) to describe various document structures (such as chapters, paragraphs, lists, etc.) had to be defined. Since that time, GML was used by many publishing systems and achieved a substantial industrial acceptance. In 1980, IBM itself, which was considered to be the world's 2^{nd} largest publisher, produced over 90% of its documents using GML.

2.1.1.4 SGML

Charles Goldfarb continued to work on improving the GML, which resulted in the creation of the Standard Generalized Markup Language (SGML) [37] in 1974. SGML quickly became accepted as a standard for information interchange and processing. The first working draft of the SGML standard was published in 1980 by ANSI. By 1983, the sixth working draft is recommended as an industry standard and adopted by such organizations as US IRS and DoD. In 1986 SGML was established as an ISO standard (ISO 8879:1986).

SGML is an extremely powerful and flexible metalanguage, however, because of its complexity very few applications could process it (and the ones that could, were generally quite expensive) and it remained a *niche* market in the 1980's, focusing

primarily on document interchange (and publishing) between large organizations.

2.1.1.5 HTML

With the creation of the World Wide Web (WWW) at the beginning of the 90s the need for the markup language that can be processed by the WWW browsers arose. Tim Berners-Lee and Anders Berglund used SGML to define a tag-based language as a means of adding meaning and presentation instructions to technical documents that were shared over the early Internet. The language was called Hyper Text Markup Language (HTML) [5] and initially had a very small set of tags (~10) that were easy to remember and use.

While HTML didn't bring any innovations to the field of markup languages, its simplicity allowed it to become extremely popular,[1] thus popularizing the ideas of document markup and enabling information interchange on a large scale.

2.1.2 XML

2.1.2.1 Origins

As the Internet grew and evolved, increasing number of companies and organizations wanted to participate in data interchange, however HTML (by far the most popular markup language at the time) was designed for a different goal (to represent how parts of the document should look, rather than what they mean), thus it turned out to be too limited for this purpose and a demand for a flexible generic markup

[1] In August 2004, Google was indexing 4,285,199,774 web pages, and this is only a part of existing HTML documents.

language emerged. SGML was powerful enough to fill this role, however it was too complex for people used to HTML (some other technical issues existed as well, e.g., it was difficult to validate over the network).

In November 1996, at the SGML'96 conference an initial draft of the Extensible Markup Language (XML) [10] was created and in February 1998 W3C accepted XML 1.0 as a standard (currently in its 3^{rd} edition [17]).

XML is a restricted form of SGML (it is a strict subset, so any XML documents are correct in SGML as well). XML has a less ambiguous syntax (e.g., all attributes must have a value, empty elements have a special syntax, etc.). Many SGML-specific features were removed, most notable of which is that a compulsory validation against a DTD [cf. Section 2.1.2.3] was no longer required (it is enough for an XML document to be *well-formed*, i.e., have a good syntax, no crossing tags, etc.). As a result the specification for XML is less than a tenth of the size of the SGML specification (overview of the differences can be found in [32]).

2.1.2.2 Documents

Every XML document is plain text and composed of nested tagged elements. Each tagged element has a sequence of zero or more attribute/value pairs and is made up of a start and end tag (there is also an alternative shortcut notation for an empty element) with data in between. This data is an ordered sequence of zero or more sub-elements. The sub-elements may themselves be tagged elements, or they may be tag-less segments of text data.

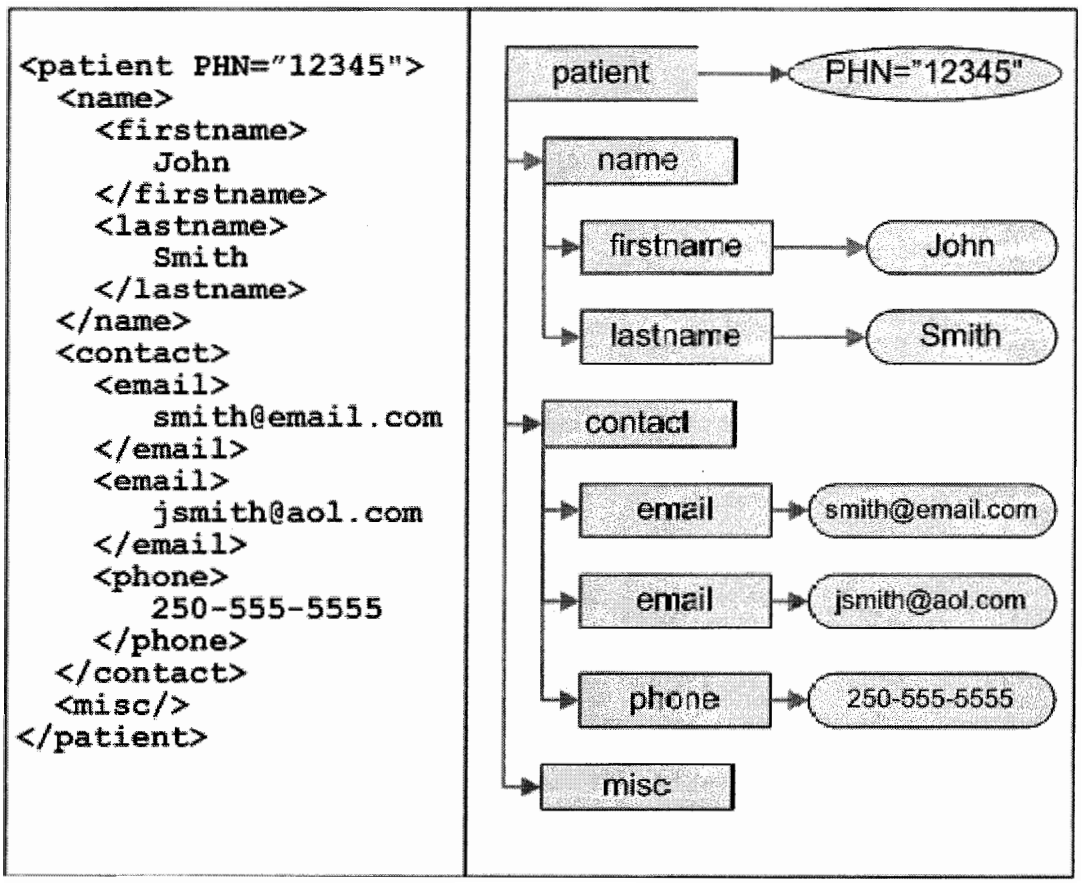


Figure 2.3. *Example of an XML document (a) and a corresponding tree (b)*

The elements, attributes and their hierarchical relationships are easily represented in a tree structure (see [2] for more details). Figure 2.3 shows an example of an XML document and its corresponding tree.

While XML is a *metalanguage*, i.e., it is designed to define other languages (or *document types* as they are called in XML), proper XML documents do not have to fit any such definitions (such documents are called *schema-less*). As long as the document conforms to the following rules, it is called *well-formed* and can be parsed

by any XML parser:

- Elements must have closing tags.
- Tags must be properly nested.
- Document must have a single *root* tag.
- Attribute values must be quoted.

2.1.2.3 DTDs and XML Schemas

While it is technically enough for XML documents to be *well-formed* and not correspond to any predefined *document type*, such documents are largely useless for the automated processing. They are still human and machine readable, however, without an agreement on what tags and attributes are allowed and in what sequence they can occur, these documents can not be verified and/or used to share data.

The list of XML markup declarations that provides a grammar for a *document type* is known as Document Type Definition (DTD) [17]. XML documents that satisfy the rules laid out in a DTD are called *valid*. Each DTD consists of declarations for elements and attributes expressed in Extended Backus-Naur Form (EBNF). Elements can nest other elements (even recursively), or be empty. Simple cardinality constraints can be imposed on the elements using regular expression operators (? for optional, * for zero-to-many, + for one-to-many). Elements can be grouped as ordered sequences (a,b) or as choices (a|b). Elements have attributes with properties type (PCDATA, ID, IDREF, ENUMERATION), cardinality (#REQUIRED, #FIXED, #DEFAULT),

```
<!ELEMENT patient (name,contact,misc)>
<!ELEMENT name (firstname,middlename?,lastname)>
<!ELEMENT firstname (#PCDATA)>
<!ELEMENT middlename (#PCDATA)>
<!ELEMENT lastname (#PCDATA)>
<!ELEMENT contact (email|phone)*>
<!ELEMENT email (#PCDATA)>
<!ELEMENT phone (#PCDATA)>
<!ATTLIST patient PHN CDATA #REQUIRED>
<!ELEMENT misc EMPTY>
```

Figure 2.4. *Example of a DTD*

and any default value. Figure 2.4 shows an example of a DTD that corresponds to the XML document in Figure 2.3a.

While simple and easy to use, DTDs also have a number of limitations (e.g., data types could not be specified in DTDs). To overcome these limitations, a language called *XML Schema*[2] [12] was proposed and became a W3C recommendation in May 2001. XML Schemas are more complex and more powerful than DTDs and can specify such document features as data types, value ranges and patterns, number of occurrences for elements, etc. The XML Schema language is also itself defined in XML and is inherently extensible. While in this work we are focusing almost exclusively on DTDs, the shift to XML Schemas is a likely future direction [cf. Chapter 7].

The use of standardized definitions for *document types* (whether using DTDs or

[2]Note that this *Schema* is spelled with a capital 'S'; *schema* with the lowercase 's' is used to refer to both DTDs and XML Schemas

XML Schemas) allows XML documents to be used in data exchange, provided all of the parties have access to such definitions. A number of WWW-based repositories (such as [8, 6]) exist to facilitate this.

2.1.2.4 XSLT (Extensible Stylesheet Language Transformations)

If the parties in data exchange share the same DTDs (or XML Schemas) then the exchange process itself is straightforward. However, if we want to exchange data that conforms to different *document types*, then the need for translation arises. Probably the most well-known language for transforming XML documents into other XML documents is XSLT (Extensible Stylesheet Language Transformations) [11].

XSLT (became a W3C Recommendation in November 1999) is a part of the Extensible Stylesheet Languages (XSL) family [3] that also includes XPath (an expression language for accessing or referring to parts of an XML document) and XSL Formatting Objects (XSL-FO, an XML vocabulary for specifying formatting semantics). XSL was designed for expressing style sheets (files that describes how to display an XML document of a given type) and its XSLT part was originally intended to perform complex styling operations (e.g., like the generation of tables of contents), however now it is used as a general purpose XML transformation language.

Just as XML was derived from SGML, XSLT originated from an SGML-based standard called DSSSL (Document Style Semantics and Specification Language) [52]. DSSSL was also originally designed to define how to render SGML documents, but can be (and is) used for general transformations of SGML documents (and since XML

is a subset of SGML, it works on XML documents as well). Aside from DSSSL, other XML transformation languages such as Omnimark [21] (proprietary language based on XML parsing events) and FXT (Functional XML Transformer) [29] (a functional language based on SML) exist as well, but XSLT is by far the most popular.

XSLT script (also called *transformation sheet*) consists of transformation rules (templates) associated with patterns of elements and attributes in XML document expressed in XPath. These patterns could be quite complex and include value checks, use predefined functions (e.g., computing the number of occurrences), etc. When a match is found in the source document, the corresponding rule is executed and generates a fragment of the target XML document. In the process, other templates might be applied (sometimes recursively) and various other side-effects (such as setting of variables) might be generated.

An example of an XSLT script and its effects are shown in Figure 2.5.

2.2 Data Heterogeneity Classification

As stated in Chapter 1, the biggest obstacle to medical data integration is the variety of ways in which similar data is represented in different data sources (or what Walter Sujansky calls *representational heterogeneity* [60]). The first layer of heterogeneity usually arises from the diversity of technologies (and/or specific implementations) of the data sources themselves. Transforming data from relational, hierarchical, object-oriented, flat file, XML-based and other types of data sources into a single represen-

a)

```
<xsl:stylesheet ...> (namespace declaration omitted)
  <!-- Copy document -->
  <xsl:template match="node()|@*">
    <xsl:copy>
     <xsl:apply-templates select="@*|node()"/>
    </xsl:copy>
  </xsl:template>
  <!-- Rename "lastname" element to "surname" -->
  <xsl:template match="patient/name/surname">
    <xsl:element name="Record">
      <xsl:apply-templates select="node()"/>
    </xsl:element>
  </xsl:template>
  <!-- Delete 2nd "email" element -->
  <xsl:template match="patient/contact/email[position()=2]"/>
</xsl:stylesheet>
```

b)

```
<patient PHN="12345">
  <name>
    <firstname>
       John
    </firstname>
    <lastname>
       Smith
    </lastname>
  </name>
  <contact>
    <email>
       smith@email.com
    </email>
    <email>
       jsmith@aol.com
    </email>
    <phone>
       250-555-5555
    </phone>
  </contact>
  <misc/>
</patient>
```

c)

```
<patient PHN="12345">
  <name>
    <firstname>
       John
    </firstname>
    <surname>
       Smith
    </surname>
  </name>
  <contact>
    <email>
       smith@email.com
    </email>
    <phone>
       250-555-5555
    </phone>
  </contact>
  <misc/>
</patient>
```

Figure 2.5. *Example of an XSLT script (a), original XML document (b), and the resulting XML document (c).* Differences between documents are highlighted.

tation is the initial step in data translation process. In our *HealthMatrix* system this step is performed by the *Service Federation Envelope*, which wraps various types of data sources and allows us for all intents and purposes to consider them XML-based [cf. Section 3.3.1]. However, even if all data sources are XML-based, significant representational heterogeneity would remain. A number of researchers (including [35, 60, 27, 30]) have worked on categorizing the types of heterogeneities. Partially based on their work, we have attempted to create a classification[3] of heterogeneity, applied specifically to XML-based data:

2.2.1 Naming Heterogeneity

This type of heterogeneity is based on the naming of data elements. It occurs when different names are used by different data sources to describe the same concept (synonyms), or when the same name is used to describe different concepts (homonyms). This type of heterogeneity is not concerned with the structure of the data elements or their values.

Naming Synonyms The same element (or attribute) can be named differently in different data sources. For example, the element **BirthDate** (from the element **Patient**) in the data source A (Figure 2.6) corresponds to the element **DOB** (from the element **Patient**) in the data source B (Figure 2.7).

[3]Note that this breakdown into categories is not the only possible one and the categories themselves are not mutually exclusive.

```
<EMR>
  <Patient>
      <PersonalInfo>
          <Name>John Smith</Name>
          <Weight>200.0</Weight>
          <Height>180</Height>
          <BirthDate>01/01/1900</BirthDate>
          <Sex>M</Sex>
      </PersonalInfo>
      <ContactInfo>
          <Phone>250-555-5555</Phone>
          <Email>john@something.com</Email>
          <Province>BC</Province>
      </ContactInfo>
      <ReferredBy>Dr. Sinclair</RefferedBy>
  </Patient>
  <CareEvent>
      <ID>12345</ID>
      <Date>01/01/2004</Date>
      <LastDate>05/05/2004</LastDate>
      <LabTest Code="PNE">Normal</LabTest>
      <LabTest Code="BP">150/100</LabTest>
  </CareEvent>
</EMR>
```

Figure 2.6. *Heterogeneity example: Data from source A*

```
<EMR>
  <Patient>
      <PersonalInfo>
          <FirstName>John</FirstName>
          <Surname>Smith</Surname>
          <DOB>January 1, 1900</DOB>
          <Sex>Male</Sex>
      </PersonalInfo>
      <Phone>555-5555</Phone>
      <Email>john@something.com</Email>
      <Email>john@another.com</Email>
      <PostalCode>V8P 3A7</PostalCode>
      <Weight>90.8</Weight>
      <Height>180.9</Height>
  </Patient>
  <CareEvent ID="12345">
      <StartDate>January 1, 2004</StartDate>
      <Date>May 5, 2004</Date>
      <LabTest Code="PNE">1.3</LabTest>
      <LabTest Code="BP">High</LabTest>
  </CareEvent>
</EMR>
```

Figure 2.7. *Heterogeneity example: Data from source B*

Naming Homonyms Two (or more) elements or attributes with the same name represent different concepts in different data sources. For example, the **Date** element from the **CareEvent** in data source A (Figure 2.6) refers to the date of the initial complaint, and thus it is different from the **Date** element from the **CareEvent** in the data source B (Figure 2.7), which refers to the date of the last visit.

2.2.2 Value Heterogeneity

This type of heterogeneity is based on the value of data elements. It occurs when the values of a particular element are represented differently in different data sources.

Numeric-Numeric If the values are numeric in both data sources, the following heterogeneities can occur:

Different units with fixed conversion Happens when different data sources use different units for the same element. For example, patient's **Weight** in the data source A (Figure 2.6) is stored in pounds, while the same **Weight** in the data source B (Figure 2.7) is in kilograms. In this case a conversion from one to the other is relatively simple.

Different units with varying conversion This is similar to the above case, but the conversion factor varies depending on time (e.g., currency), geographical area (e.g., provincial tax rate) or other parameters.

Same units with different precision This form of heterogeneity occurs when the same data is stored in different data sources with different precision. For ex-

ample, patient's **Height** in the data source A (Figure 2.6) is rounded to the nearest centimeter, while the same **Height** in the data source B (Figure 2.7) is stored to the nearest tenth of a centimeter.

String-String The following heterogeneity types can occur if the values are represented as strings in both data sources.

Value synonyms Occurs when a different set of string values is used by different data sources, even though the meaning of these values is the same. One of the most common examples would be the use of **'M'** and **'F'** in the in the data source A (Figure 2.6) and **'Male'** and **'Female'** in the data source B (Figure 2.7) to define the patient's sex.

Value homonyms Occurs when the same string value has different meaning in different data sources. For example, **LabTest**'s **Code** with the value 'PNE' in data source A (Figure 2.6) represents 'Pneumonia', while the same value in the data source B (Figure 2.7) represents 'Pneumoconiosis'[4].

Note that these two cases (*value synonyms* and *value homonyms*) are especially common in the medical domain. As Walter Sujansky says in [60]:

> *In the biomedical domain, where nomenclature is complex, sometimes ad hoc, and often overlapping, this vocabulary problem is a significant issue for any system that seeks to aggregate or compare data collected*

[4]Example taken from [35].

at distinct sites.

This issue is so important that it is dealt with by a whole subfield of medical informatics and large government-funded terminology resources, such as Unified Medical Language System (UMLS) [24] and Logical Observation Identifiers Names and Codes (LOINC®) [18].

Different formats Occurs when the same string value is stored in different format by different data sources. This is very common for the time and date values. For example, data source A (Figure 2.6) stores dates in the "DD/MM/YY" format, whereas data source B (Figure 2.7) stores them as "Month DD, YYYY".

String "precision" Occasionally the same string data is stored by one data source in a less "precise" (i.e., a prefix/suffix that is assumed to be standard could be omitted, words shortened, etc.) form than in the other. For example, the **Patient**'s **Phone** is stored with area code in data source A (Figure 2.6), but without it in data source B (Figure 2.7).

Numeric-String This type of heterogeneity occurs when the same value has a different data type in different data sources. It is relatively rare for XML documents, because the majority of existing documents are based on DTDs [cf. Section 2.1.2.3] and treat all values as strings. However, we can expect it to occur more frequently with the growth of popularity of XML Schema [cf. Section 2.1.2.3].

2.2.3 Content Differences

This type of heterogeneity occurs when data represented in one data source is not directly represented in another. Such data may be implicit, derivable, or simply missing.

While in some cases (e.g., *implicit data*) it is similar to *value heterogeneity* [cf. Section 2.2.2], conceptually it is quite different, because it deals with the data that is not represented at all, rather than represented differently. Also note that this type of heterogeneity doesn't necessarily apply uniformly across a data element (e.g., all phones are NULL in one data source, but have correct values in another), but could be present just in some instances of stored data (e.g., John Smith's phone number is stored in one data source, but is missing from another).

Implicit data Implicit data is usually constant, and therefore assumed, within the environment of a single data source, but cannot be assumed across different data sources. For example, data source B is local to Victoria, BC, so it implicitly assumes that the *area codes* for all **Patient**'s **Phones** (Figure 2.7) is *250*, while data source A (Figure 2.6) makes no such assumptions.

Derivable data Derivable data is the data that, while not directly represented in a data source, can be inferred from other data elements. For example, data source A (Figure 2.6) stores **Patient**'s **Province**, while data source B (Figure 2.7) contains **Patient**'s **Postal Code** instead. Each one of them can be derived from the other,

albeit sometimes (in case of **Province→Postal Code**) with a lack of precision.

Missing data Occurs when the data is simply not stored in one of the data sources. For example, a general clinical facility might omit the patient's mental status information (out of privacy concerns), while a psychiatric facility would provide such data. Missing data is usually stored as *NULL*, however it should be noted that, while common, this practice is deficient, because the semantic of *NULL* is ambiguous [33] [cf. *Meaning of NULL*, Section 2.2.4].

2.2.4 Semantic Heterogeneity

Since there is no agreement in the field about the definition of *semantic heterogeneity* [57], we have decided to use a narrower one from *El-Khatib et al.* [35]:

> *This form of heterogeneity occurs when there are differences in what the data actually represents or the context in which the data has been captured in different databases.*

The broader definition could, for example, include the types of heterogeneity that we have classified as *naming heterogeneity* [cf. Section 2.2.1] or *value heterogeneity* [cf. Section 2.2.2]

What the data represents Occurs when the same concept has (possibly) a different meaning in different data sources. For example **Patient**'s **Phone** in data source B (Figure 2.7) is a *home phone*, whereas **Phone** in data source A (Figure 2.6) is a

contact phone, which might be the same phone, but not necessarily so.

Context in which data is captured is very important and might influence the data considerably. This is especially true in the medical domain. For example (from *El-Khatib et al.* [35]:

> *If blood pressure is measured at home by a nurse the measurement may be significantly lower than that obtained in the clinic by a doctor (so-called 'white coat' syndrome). Equally one would like to know whether a measurement may be affected by other conditions (e.g., if a patient being examined for condition X is also suffering from condition Y at the same time).*

Data granularity Different data sources might capture the data with various granularity (Note: this is different from 'precision' [cf. Section 2.2.2]). For example, *blood pressure* (**LabTest** with attribute **Code="BP"**) is stored as a numeric value in data source A (Figure 2.6), but is mapped to a more abstract scale of {Very Low, Low, Normal, High, Very High} in data source B (Figure 2.7).

Meaning of *NULL* Meaning of *NULL* (or "0", "", etc.) can vary between different data sources or even within a single source. For example, if we consider patient's HIV status, *NULL* can mean *negative*, *unknown* (e.g., test has not been performed) or *known, but unavailable* (e.g., if it has been omitted for privacy reasons). In other

cases *NULL* might have even more meanings, such as *not applicable* (e.g., ovarian cancer data for a male patient).

2.2.5 Data Model Heterogeneity

These types of heterogeneity are based on the policies of the organizations that obtain and store the data. They are often hard to resolve, because they are rarely specified in the schemas. Usually, the help of the domain expert would be required to find and reconcile such heterogeneities.

Storage policy differences Occurs when different data sources have different policies regarding the amount of data that is stored by the system or the lifetime of such data. For example, one organization might store information about all visits for a particular patient, while another one might only store such data for the last 10 years.

Differences in constraints Occurs when the data elements (or attributes) are under different constraints in one data source than in another. For example, if a medical organization specializes on a particular age group (e.g., children), then the **Patient's BirthDate** in that data source would be constrained to dates consistent with that, while another data source would not have such constraint.

2.2.6 Information Capacity Heterogeneity

This type of heterogeneity arises when one of the data sources is capable of storing more extensive information than another one. The two main ways in which this can

occur are:

Missing elements One or more elements that occur in one data source are missing in the other. By 'missing' we mean that they do not occur in the data source at all, not just that they do not occur at the same position ([cf. Structural Heterogeneity, Section 2.2.7] or that their values are missing [cf. Content Differences, Section 2.2.3]. For example, the **ReferredBy** element from data source A (Figure 2.6) is missing from data source B (Figure 2.7)

Different cardinality Occurs when a relationship between two elements in one data source has a different cardinality than the same relationship in another data source. For example, the **Patient** in data source A (Figure 2.6) has only one **Email** (1:1 relationship), whereas in data source B (Figure 2.7), he/she has many **Email**s (1:n relationship). Unlike many other types of heterogeneity, this represents only a *possible* conflict in actual data, because if the patient has only one email address, then the XML documents would be identical even if the schemas differ.

2.2.7 Structural Heterogeneity

These types of heterogeneity are based on the structure of XML elements, rather than the data they contain. They occur when the elements containing the same information have different structure in different information sources.

Element/Attribute Occurs when a particular concept is represented by a leaf element in one data source, but as an attribute in another. For example, in data source A (Figure 2.6) **CareEvent**'s **ID** is an element, whereas in data source B (Figure 2.7) it is an attribute.

Single element/multiple elements Occurs when the same concept is represented by a single element in one data source, but is divided into several elements in another. For example, patient's name in data source B (Figure 2.7) is composed from **First-Name** and **LastName**, whereas in data source A (Figure 2.6) it is stored in a single element **Name**. Another example, which is very common in medical domain is the separation (or concatenation) of laboratory result values and units (i.e., <145 mg> versus <145, mg>)

Aggregation conflict Occurs when two non-leaf elements (representing the same concept) from different data sources have different sets of child elements. For example, unlike the data source B (Figure 2.7), **PersonalInfo** in data source A (Figure 2.6) includes patient's **Height** and **Weight**.

Generalization conflict Occurs when several related concepts that were separate in one data source have been grouped together as children of a generalized concept in another data source. For example, separate elements **Phone** and **Email** in data source B (Figure 2.7) are subelements of the generalized concept **ContactInfo** in

data source A (Figure 2.6).

All of these subtypes of *structural heterogeneity* could be (and usually are) combined in various ways within a data source, thereby producing drastic structural differences between data sources containing the same information.

2.3 Graph Transformations

"Graphs are very suitable for describing complex structures in a direct and intuitive way, and for this reason they are widely used in many fields of computer science. [56]" Typically, nodes represent objects or concepts, and edges represent relationships among them. Additional information is expressed by adding attributes to nodes or edges. Examples of the use of various graph-based representations range from Unified Modeling Language (UML) [15] diagrams in software engineering to entity-relationship [36] diagrams in databases and Petri nets [48] in modeling and analysis.

Given the widespread use of graphs for data representation, it is natural that graph transformations form the basis of many useful computations. Graph transformations can be represented implicitly (embedded in a program that constructs or modifies a graph) or explicitly (as graph rewriting rules that modify a graph). The explicit use of graph rewriting rules provides an abstract and high-level representation of a solution.

A graph rewriting rule is applied to a host graph to replace one subgraph by

another and usually consists of two graphs (called left- and right-hand side) and an *embedding description* [45] (that specifies how to attach a new subgraph to the host graph and might be implicit for some formalisms) and two optional components:

Application condition: Specifies when the rule can be applied. Can include restrictions on the existence of nodes and edges as well as on attribute values. Sometimes it is embedded into the left-hand side graph.

Attribute transfer function: Specifies how to assign the attribute values to the resulting graph based on the original attribute values. Sometimes it is embedded into the right-hand side graph.

When the graph rewriting rule is applied to a host graph:

1. the rule is executed only if the application conditions are met,
2. the subgraph isomorphic to the left-hand side (LHS) graph is removed from the host graph,
3. the right-hand side (RHS) graph is connected to the resulting graph, conforming to the embedding description, and
4. new attribute values are computed by the attribute transfer function.

In Figure 2.8 we show an example of graph rewriting. The rule itself is specified graphically in Figure 2.8a and it's goal is to assign a patient named Bob (to simplify the example; realistic example of this rule would have the patient's name as a parameter) to the doctor. Application conditions are embedded into the LHS graph and

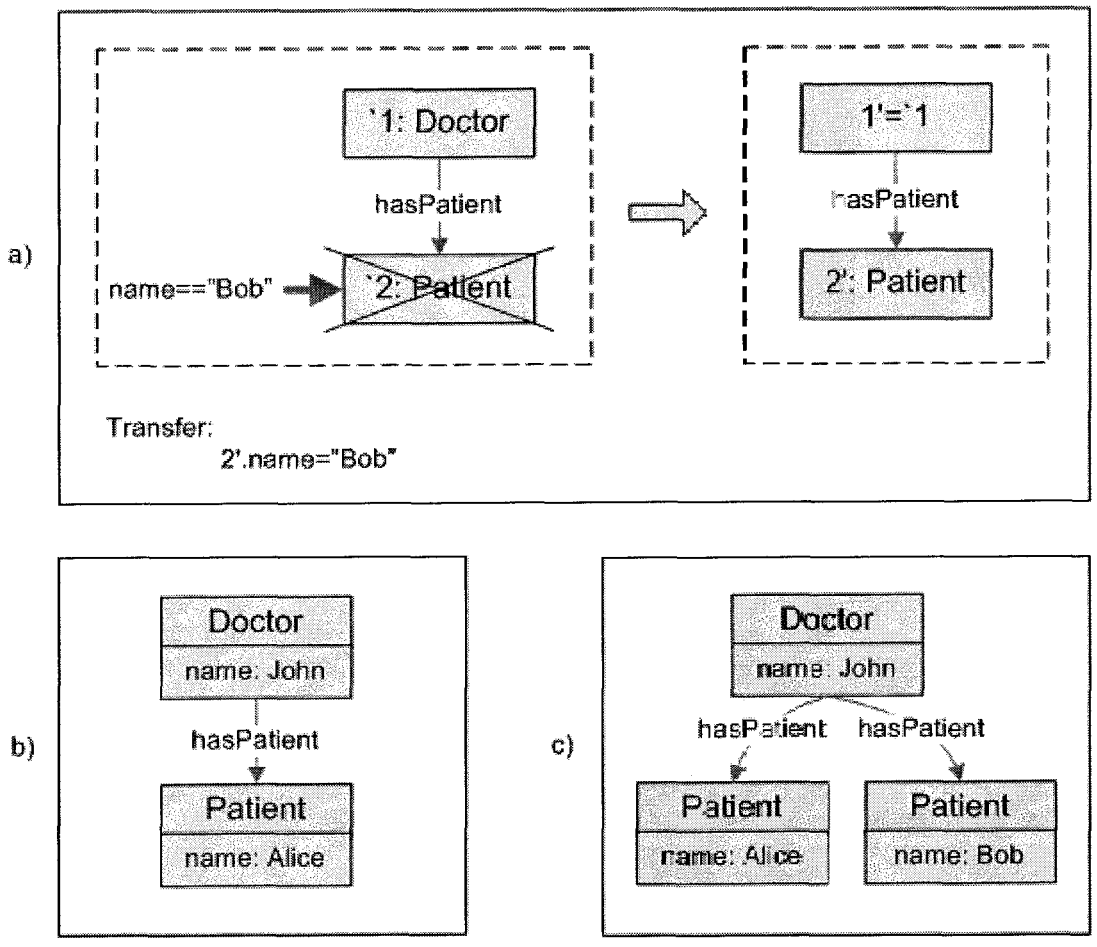


Figure 2.8. *Example of the graph rewriting. a) Graph rewriting rule. b) Original graph. c) Resulting graph.*

restrict both nodes and attributes. They specify that the rule can only be applied if the *doctor* doesn't have (shown as a crossed-out node) a *patient* with name Bob (shown as a condition attached to the *patient* node). Once the LHS is matched to the host graph (Figure 2.8b) it is replaced with the RHS. The *embedding description* is embedded into the RHS and specifies that the node '1 is replaced with an identical node ('1=1') and that node 2' and edge between 1' and 2' is created. The *attribute transfer function* specifies that the *name* attribute of node 2' should be set to "Bob". The resulting graph is shown on Figure 2.8c.

Of particular interest to us is that the graphs provide a very useful and easy to understand method of depicting both the schemas [cf. Section 5.1] of XML documents and the instances [cf. Figure 2.3] of the documents themselves. In the same fashion, we can use graph rewriting rules to represent the transformations that we propose to apply to such schemas and documents in order to resolve the data heterogeneity problem.

Chapter 3

HealthMatrix Health Information Grid

3.1 Introduction

Over the last several years, the increased availability (and reduced cost) of broadband Internet connectivity in both organizations and households caused a growing number of government agencies and businesses to consider using distributed technology to improve quality of their services and offer new services that were impossible or impractical before. Health services is one of the areas that can especially benefit from employing *grid technologies* [28] because of its inherently distributed nature involving multiple organizations and data sources (e.g., labs, hospitals).

Several reports [41, 42, 40] stated Canadian Health system's interest in such information technologies as *key enablers in meeting the challenges of the 21st century and the shared commitment to deploying them within the health sector in Canada* [42]. This interest has been further confirmed in November 2002 in a report by the Commission on the Future of Health Care in Canada (also commonly referred to as "Ro-

manov's report") that recommends *developing a pan-Canadian electronic health record framework* [55] that will allow Canadians *access to their personal health records* (parts of which could be stored at different locations) and will help medical organizations by providing them with more comprehensive and up-to-date information.

It would be impractical to disregard existing *Medical Information Systems* (MIS) and to build such a framework from scratch, so the better solution would be to create a middleware that can facilitate data and service interchange between existing MISs. This framework (further refered to as *Health Information Grid* (HIG)) has a number of important requirements:

- The more medical organizations join the grid, the better services it will be able to offer, thus HIG has to allow easy integration of new MISs and support large-scale network of components [cf. Section 3.3.2] to manage the data/control flow (and allow rapid and asynchronous evolution of such network)
- The organizations federated by the HIG will most likely have their own pre-existing MISs, so the grid has to be able deal with systems heterogeneity.
- HIG deals with health-related data, so *privacy*, *security* and *accountability* (in the form of *audit trails*, for example) are of paramount importance.

In the remainder of this chapter, I will describe the *HealthMatrix* project that was developed by our research group to address the above concerns. Section 3.2 gives more detailed requirements for our *Health Information Grid*, Section 3.3 outlines the *HealthMatrix*s architecture, and Section 3.4 describes the interactions between

components of the *Health Information Grid*.

3.2 Requirements for the Health Information Grid

Scalability: *Health Information Grid* should be able to mediate Medical Information Systems on different scales: from relatively small (within a single organization) to a very large (for a pan-Canadian system).

Adaptability/Adaptiveness: Since most of the organizations participating in HIG will have pre-existing *Medical Information Systems*, the grid has to accommodate a multi-factor (e.g., data representation, technical parameters, legal issues) heterogeneity. In order to do that HIG has to employ a human-driven customization (*adaptability*) as well as an automatic adaptation (*adaptiveness*).

Evolvability: The "layout" of health service providers mediated by the grid is dynamic. Organizations could join the grid or exit it, services could be added or removed. The HIG has to support the easy evolution of the network in two main directions: configuration of the grid (i.e., the number of components and connections between them) and federation of the new (or changes in the existing) *Medical Information Systems*.

Activeness: Due to the large scale, need for easy evolution and highly decentralized nature of the *Health Information Grid* the middleware has to be *active*, i.e., the individual components comprising the grid [cf. Section 3.3.2] should be able to pull data from the organizations, route it, transform it, push it to the

organizations and not just react to external service requests.

Security/Privacy *Health Information Grid* deals with personal medical information, so ensuring *privacy* and *security* is vital for its success. Only authorized individuals (or components) should be able to view/modify parts of data for which they have patient's consent. For example, both patient's physician and psychiatrist should be able to access his blood test data, but only the psychiatrist should see the psychiatric record.

Accountability: In order to ensure that the *privacy* and *security* requirements are fulfilled, it is important to have a mechanism that can be used to trace the sensitive data and find out where this information was distributed and who accessed and/or modified it. This mechanism should be available to both government oversight organizations (especially in the event of a security breach) and to the individuals (in order to trace and possibly withdraw their private data). Keeping *audit trails* is one of the most widely used methods to ensure *accountability.*

Reliability: Due to the nature of data transported by the *Health Information Grid* it is crucial that it doesn't get lost or corrupted. Therefore *reliability* is very important and single points of failure can't be tolerated. Performance of the HIG should degrade gracefully when individual components become unavailable.

3.3 *HealthMatrix*s Architecture

The *Health Information Grid* architecture consists of three main concepts called *Service Federation Envelope* (SFE), *Adaptive Process Middleware* (APM), and *Medical Exchange Agency* (MEA)

3.3.1 Service Federation Envelope

As mentioned above, the *Health Information Grid* federates *Medical Information Systems* that are heterogeneous on many different levels. The *Service Federation Envelope* (SFE) is used to wrap the access to participating MISs in order to resolve most of these issues:

1. Organizations use different software to build their information systems. SFE has a plug-in architecture to accommodate the so-called ImpEx (import/export) conduits, that can interface with various DBMSs. Our group has implemented conduits for MS SQL Server, MS Access and Oracle 9i, but, if another type of IS has to be federated, it is relatively easy to create an ImpEx for it.
2. In order for the middleware to function, information has to be translated between the native format of each MIS and the grid's common format. Due to the specific nature of health information, a number of *document types* (described in more details in Section 3.3.2.1) is defined in the *Health Information Grid*. SFE administrator can specify mappings between elements of the wrapped MIS and elements of a particular *document type*. Whenever a specific document has to

be send by the SFE, the relevant data is retrieved from the MIS and converted to XML (because of its ease of use and flexibility, XML is ideally suited for the common format).

3. HIG can use these mappings (see item 2.) to process the queries against native data content.

4. As all of the other components of the HIG, SFE uses Web Service technology to provide a standardized interface.

5. *Note: In case if the native schema of the MIS is too different from the required document additional translation with the help of the Transducer component [section 3.3.2.4] may be required.*

A more detailed description of the SFE architecture is beyond the scope of this thesis, but it can be found in [50].

3.3.2 Adaptive Process Middleware

Adaptive Process Middleware transports information (in the form of *tokens* [cf. Section 3.3.2.1]) between medical organizations federated by SFEs. Each APM network consist of instances of customizable components (from a predefined set) linked into a P2P network [cf. Figure 3.3]. Most types of APM components are active, i.e., they can pull a *token* from a *Staging Area* [cf. Section 3.3.2.2], perform necessary manipulations on it and then push the *token* to another *Staging Area.*

The business processes supported by a *Health Information Grid* define the con-

figuration of its APM network. At the moment this process is human-driven, i.e in order to add or change a supported process someone has to redesign the network using an APM Admin tool. In the future, we hope to make the APM adapt to the new requirements automatically.

The remainder of this section describes various types of components that make up the APM network and format of the transported data. The interaction between components and theoretical foundations of the *Adaptive Process Middleware* is explained in detail in Section 3.4.

3.3.2.1 Documents and Tokens

Information units transported over the APM network are called *tokens*. Each token's "payload" is a medical document of a particular type (for example *Blood Test* or *Electronic Medical Record* (EMR)). Possible *document types* are pre-defined and organized into a hierarchical structure [cf. Figure 3.1] based on the *Clinical Document Architecture* (CDA) [34].

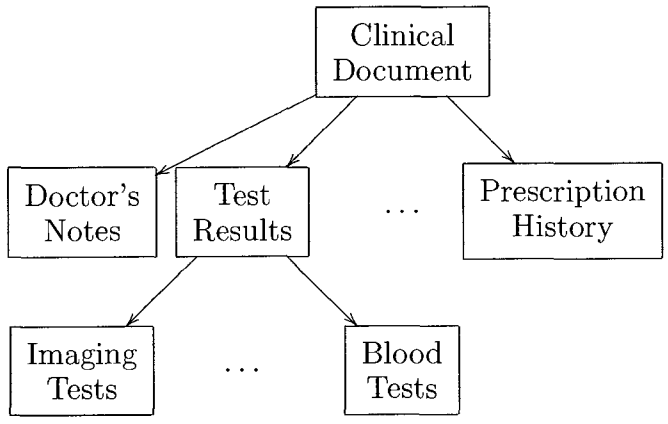


Figure 3.1. *Sample Document Hierarchy*

Documents produced by the SFE contain only the data that it's MIS has (or chooses to make available) for a specific *document type*, so two documents with the same type produced as a result of a same query by two different SFEs might not contain the same information.

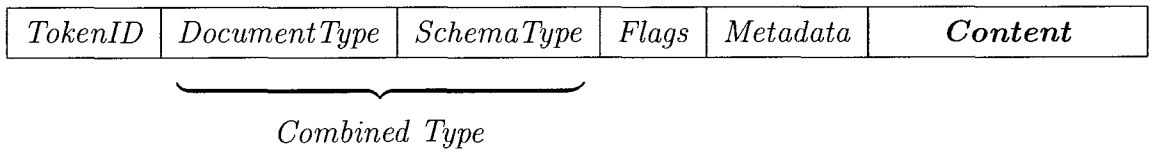


Figure 3.2. HealthMatrix *Token Format*

HealthMatrix tokens are encoded in XML and consist of the following parts [cf. Figure 3.2]:

TokenID: Unique ID used by MEA [cf. Section 3.3.3] to track the token.

Document Type: Indicates type of information (from within the document hierarchy) that the token is carrying. e.g., Blood Test token or EMR token.

Schema Type: Indicates the schema used by token's data. Most SFE's would use HL7 RIM schema [4], but sometimes it is necessary to allow SFE's to generate tokens with non-RIM-compliant data.

Note: A combination of document type *and* schema type *is called a* combined type *(also sometimes referred to as* colour *of the token).*

Flags: Used to define the class of token (e.g., query token). Differences between token classes are described in details is Section 3.4.2.3.

Metadata: Used by APM components to store routing data and other information

internal to APM network.

Content: Clinical document or query that the token is carrying. *Content* is XML encoded and usually encrypted.

3.3.2.2 Staging Area

The only passive component of the *Adaptive Process Middleware*. Provides temporary storage for the information units before they are processed by the next active component. *Staging Areas* allow other components to perform asynchronously and concurrently and improve the reliability of the APM network (if one of its target components is temporarily unavailable, *Staging Area* can hold the *token* until that component becomes available again).

3.3.2.3 Initiator

The purpose of the *Initiator* is to react to the external events (in the form of Web-Service calls) and generate *query tokens* that are routed [cf. Section 3.4.2] over the network of APM components to the appropriate organizations in order to retrieve data. The *query tokens* are created from the *parameterizable query templates* that are designed by domain experts using an external tool (Visual Query Editor) and stored in the *Medical Exchange Agency* [cf. Section 3.3.3]. If one wants *Initiator* to support another query, new templates could be easily downloaded from MEA, allowing on-the-fly reconfiguration.

3.3.2.4 Transducer

Translates information units from one *schema type* [cf. Section 3.3.2.1] to another. *Transducer* uses predefined translations scripts created by an external tool and downloaded from MEA. Like all other components that use scripts (see Sections 3.3.2.3, 3.3.2.5), it can be reconfigured on the fly.

The *translation specification tool (Odin)* and *Transducer* (as well as the theory behind them) are the main focus of this thesis and described in details in the following chapters.

3.3.2.5 Merger/Adder

Combines several information units of a different structure (but storing the same type of data) into a combined information unit. Uses predefined scripts (downloaded from *Medical Exchange Agency*) to facilitate more complex merging.

Adder is a simpler version of *Merger* and can combine only information units with the same structure.

3.3.2.6 Workflow Execution Engine (WEE)

The *Workflow Execution Engine* differs from all other APM components by the fact that it can only be located at the federated MIS. In a way it behaves more like an additional feature of the SFE then an APM component. WEE executes predefined guidelines that are created with the help of an external tool and could be downloaded from MEA. In the process of executing a workflow, WEE can enrich tokens with

additional information, call for additional tokens, send tokens to other SFE in order to request data or services, make clinical decisions automatically (in simple cases) or ask for human intervention (in complex cases).

3.3.3 Medical Exchange Agency

The *Medical Exchange Agency* (MEA) serves as both the central repository and a main control/administration point for the *Health Information Grid*. It has the following major purposes:

Configuration: MEA stores the configuration information for the APM network. This information contains all components of the network, links between them and the *document types* that they can produce/consume.

Certification: Every component of the APM network has to register with a MEA. In order to ensure *security* and *privacy*, an APM component can function (consume and produce data) only if it is currently certified by MEA. This allows MEA to withdraw it's certificate and prevent component's access to confidential information if an organization leaves the grid (or if a component is compromised). In order to avoid single point of failure, APM components can continue to function for a predefined period of time if MEA is temporarily unavailable.

Auditing: In order to fulfil the *accountability* requirement [cf. Section 3.2] whenever a *token* with personal information (the MIS's elements that are considered to be personal data are flagged during the setup process of each SFE) leaves or enters

a SFE, the complete *audit trail* (containing that personal information, the full path of the *token* through the APM network and points of it's modification) is sent to the *Medical Exchange Agency*. The MEA provides a repository to store the *audit trails* and a mechanism to search them.

Scalability: *Health Information Grid*s could be connected together to create larger HIGs. For example, internal grid for a hospital can be joined with grids from other hospitals to form a city-wide HIG. MEAs support this by providing interface to the subnet and serving the same role in the higher level network as a SFE would.

Knowledge Repository: MEAs also serve as repositories for the common information used by multiple APM components. They can store scripts for Transducers [cf. Section 3.3.2.4] and Mergers [cf. Section 3.3.2.5], practice guidelines for Workflow Execution Engines [cf. Section 3.3.2.6] and other similar data.

3.4 APM Network

3.4.1 Visual Representation

Even though *HealthMatrix* has only six different types of components, it is possible to construct very complex networks with them. We use a graph-based visual language to make it easier to define and view APM networks. Figure 3.3 shows a sample *Health Information Grid*. The symbols representing each component [cf. Section 3.3.2] are explained in Table 3.1.

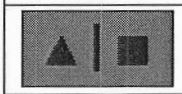
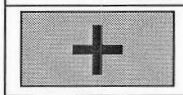
Symbol	Component
	Organization federated by an SFE
	Staging Area
	Transducer
	Merger
	Initiator
	SFE with a WEE attached

Table 3.1. *Visual representations of* HealthMatrix *components*

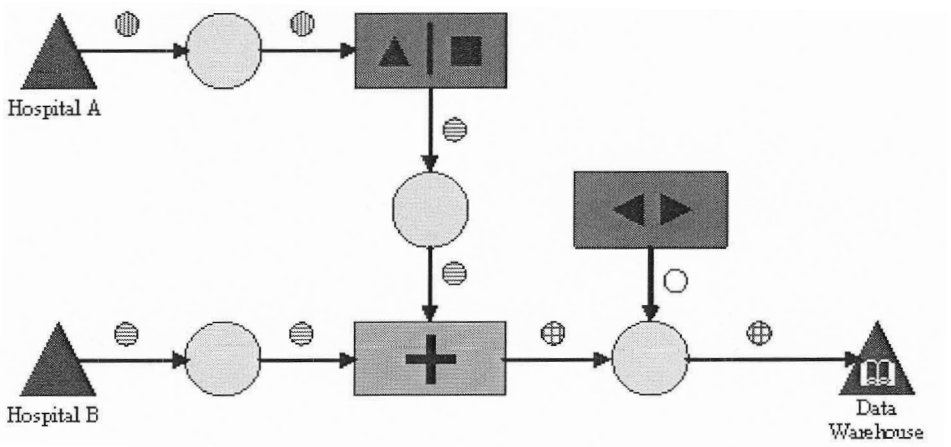


Figure 3.3. *Sample Health Information Grid*

The arcs show connections between components and their direction corresponds to the directions in which most tokens (*query tokens* flow in opposite direction) travel through the APM network. In order to control the information flow, the arcs are typed and labeled with the *composite types* (visually represented as colored circles) [cf. Section 3.3.2.1] of tokens that are allowed to go through them.

Figure 3.3 shows a simplified example from our Palliative Care case study. The goal of this sample network is to collect the data from several hospitals into the data warehouse. After the *initiator* sends a query token, this token traverses the network (against the direction of the arcs) and causes the federated medical information systems to send the tokens with requested data. After the data from *Hospital A* is translated into a common format it is merged with data from the *Hospital B* and sent to the *Data Warehouse*, where it can be analyzed (with the help of the WEE).

3.4.2 Component Interaction

Since our *HealthMatrix* middleware project deals with critical information, having a formal model for it is very important, since it will allow us to reason about requirements defined earlier in this chapter and other properties of the APM networks. The formalization of the interaction semantics of the *Health Information Grid* components is based on the *colored Petri nets* [46, 48]. The formal analysis of the requirements and the mapping between APM network graphs and *colored Petri nets* is not included in this thesis, however, the informal description of *Petri net* execution rules (as they

apply to APM networks) is shown in Section 3.4.2.2. In the remainder of this section we describe the interaction between APM components.

3.4.2.1 Component Connection Rules

1. Active APM components have to be provided with buffers for their input/output information, so they can only be directly connected connected to *Staging Areas* and not other active components.
2. Since *Transducer* and *Merger/Adder* both consume and produce tokens, they must have at least one incoming and one outgoing *Staging Area.*
3. *Initiator* doesn't consume tokens, so it only has to have outgoing *Staging Areas.*
4. SFEs could be token producers, consumers or both, so they are not required to have both incoming and outgoing *Staging Areas*, but must have at least one of them in order to be connected to the network.
5. *Workflow Execution Engine* can only be located at the SFE, so it follows SFE's rule.

3.4.2.2 Component Execution Rules

1. An incoming place of an active component is connected to it with the arc that is labeled with a set of *composite types* [cf. Section 3.3.2.1]. When the incoming place has received enough tokens with the right *composite types* to match the arc labelling, that arc becomes enabled.
2. When all incoming arcs on an active component are enabled then this compo-

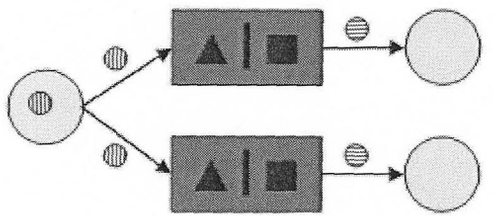


Figure 3.4. *Example of the Competition for Tokens*

nent becomes enabled as well and can perform its function (of *fire*).

3. When the component becomes enabled, it is not required to fire immediately and can decide for itself when to run.

4. The enabled component can become disabled again if one (or more) of it's incoming tokens becomes unavailable (consumed by another component, expire, etc.).

5. When the component fires, it consumes all of it's incoming tokens (though not necessarily all tokens in it's incoming staging areas) and produces appropriate tokens (corresponding to the labelling of it's outgoing arcs) in it's target *Staging Areas*.

   Note: There are some exceptions to this rule specified in Section 3.4.2.3.

3.4.2.3 Behavior of the different token classes

As it was shown in Section 3.3.2.1, there are several classes of tokens with somewhat different behaviors:

Transaction Tokens: Usually represent resources or services. Sometimes components have to compete for them. For example, in Figure 3.4 both *Transducers*

are enabled, but only one can fire, because whichever one fires first will consume the token and disable the other *Transducer*. Negotiation for the right to consume that token is based on a number of criteria (depending on the *document type* of the *transaction token*).

Information Tokens: Represent information transmitted by SFE. There is no competition for the *information tokens*. As soon as it is consumed by a component, the *information token* will respawn and allow other components to consume it. *Staging Areas* keep track of the respawned tokens to prohibit their multiple consumption by the same components. *Information tokens* might have an expiration timer and disappear by themselves if not consumed within certain time frame.

Query Tokens: Used to get information from SFEs. *Query tokens* traverse the grid in the opposite direction to all other tokens. Since all queries are represented using HL7 RIM schema (or subset of it) and are used to query for a specific *document type* (regardless of schema that the queried SFE will use to encode it), their flow is based only on *document types* part of the arc labelling. All components except the SFEs ignore the query tokens and don't process them.

Directed Tokens: Special subclass of both *transaction* and *query tokens*. These tokens are targeted to a specific SFE and are routed by the network accordingly.

Chapter 4

Translation of XML Data

4.1 Motivation

As we have shown in previous chapters, there is a very large (and quickly growing) number of XML documents available. In many cases these documents contain very similar data, but conform to heterogenous schemas. Such heterogeneity arises from the fact that they come from the sources that were designed through a largely decentralized process intended to meet specific (or local) data needs without any coordination between organizations that provide this similar data. However, whether these documents were produced in XML originally or as a result of a database being wrapped by a SFE [cf. Section 3.3.1], their set of allowed elements (and attributes) along with the valid structure is specified by the DTDs (or XML Schemas).

In order for anyone to use this *latent* information provided by other organizations, first it has to be translated into an understandable form. For a long time (and largely today as well), programs for such translations were manually created in languages such as XSLT[1] [cf. Section 2.1.2.4].

[1]Other transformation languages such as DSSSL, Omnimark, and a variety of experimental ones

An XSLT program (alternatively called stylesheet[2] or script) is usually created after careful and detailed analysis of semantics and structure of both source and target XML schemas (and often instances of XML documents as well) in order to discover the differences and similarities between them. After this analysis is completed, the XSLT program is manually written. However, this process is very time-consuming and error-prone in both the analysis and coding phases:

Analysis: Both the source and target schemas can be very large and complex, thus it is quite easy to overlook important details. In addition to that, schemas for the documents in some domains (such as healthcare) would contain extensive domain-specific terminology and requires domain-expert as an analyst.

Coding: XSLT is a very powerful transformation language, however, it is also very complex and even a simple transformation might result in an elaborate XSLT program. Furthermore, because of some properties of XSLT, even a minute and incremental change of one of the schemas might require a complete rewriting of the XSLT program.

Another important issue for data translation in domains such as healthcare, that is largely overlooked in the manual analysis/coding process, is that of reasoning about how the critical and/or sensitive data is modified by the translation. For example, if we want to anonymize the EMR (i.e., strip all identifying information about the

exist as well, but they are much less popular [cf. Section 2.1.2.4].

[2]This name originated from the initial use of XSLT as a part of XSL to present XML data as XHTML for the web browsers

patient from it), it is important to be able to prove that this information can not be restored. The same concern (albeit in the opposite direction) applies to being able to show that the critical information is not lost in the translation process.

4.2 Existing Approaches

To simplify the creation of XML document translators in the last several years a number of approaches ranging from new translation languages to the visual editors have been developed. These approaches are mainly concentrated on tackling these two issues: *schema matching* (i.e., how to determine the corresponding elements between two schemas) and *translation specification* (i.e., how to simplify the coding of the XML document translators).

4.2.1 Schema Matching

As mentioned in Section 4.1, before the actual translation program is written, mappings between the two XML schemas have to be determined. The traditional approach, where the human expert analyzes the schemas (possibly with the help of one of many schema visualization applications) and determines these mappings is often quite complex, time-consuming and error-prone, especially if the schemas are large and substantially different. An alternative to this approach is to employ an automatic (i.e., the mappings are determined without any human intervention whatsoever) or semi-automatic (i.e., when the application guides and/or assists the human expert)

schema matching.

Schema matching is not a new problem and has been the focus of database community for some time. A wide variety of approaches to automatic schema matching exists and have been classified [cf. Figure 4.1] by Rahm and Bernstein in [54]:

Schema-based / Instance-based: Schema-based matchers consider only schema information (i.e., elements, relationships between them, cardinality, etc.). Instance-based mappers also use statistical information about actual data or sample documents.

Element granularity / Structure granularity: Element-level matchers only compare individual elements (disregarding the structure of the schema), whereas structure-level mappers analyze structures in which elements appear in both schemas.

Linguistic analysis: Linguistic matchers compare the similarities of names of the schema elements and attributes. They can use various dictionaries and domain-specific terminology resources in order to resolve the synonyms and homonyms.

Constraint-based: Constraint-base matchers use constraints on the schema elements, such as data types, value ranges, cardinalities, etc.

Many automatic schema matchers use a combination of these approaches (whether evaluated independently or in combinations) and occasionally augment them with some sort of machine learning technique.

Even with the assistance of all these approaches, automatically discovering se-

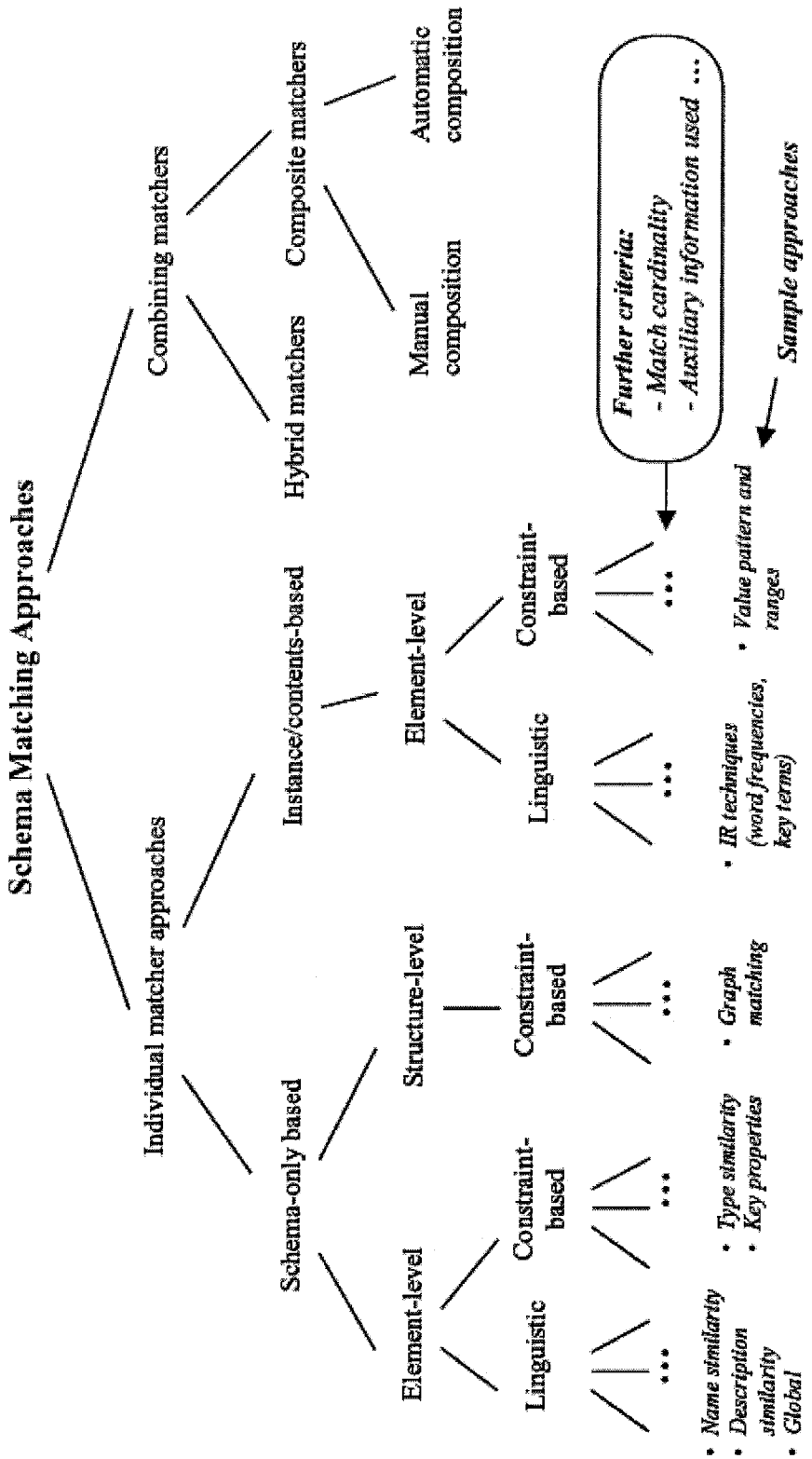


Figure 4.1. *Classification of schema matching approaches from [54].*

mantic correspondences between heterogenous schemas is very difficult, since these schemas can be drastically different in a variety of ways [cf. Section 2.2 for a detailed description]. Most of the automatic schema matching approaches also share a number of fundamental limitations:

- Almost all automatic matching algorithms measure similarity between schema elements. This similarity is usually expressed as a value in [0,1] (sometimes multiple metrics are used, resulting in vectors of such values). While this is a very convenient mathematical abstraction, it is often hard to accurately map a similarity of 0.8 (for example) between two elements into a semantic relationship between these elements (and especially into an operation that has to be used for the translation between them) [30].
- While most of the automatic schema matchers can discover 1:1 matches between schema elements, a majority of them is not very adept (and quite error-prone) [62] in detecting n:m matches (e.g., when a full name in one schema is broken into first/middle/lastname in another).
- Many types of heterogeneities (such as *value heterogeneity* [cf. Section 2.2.2]) are concerned with the values of the data, rather than with the schema. Most of automatic schema matchers analyze only the schema and could not even detect it. Even if the schema matcher is instance-based, it would only be able to resolve some subtypes of *value heterogeneity*, such as different data formats or possibly *value synonyms* (and even that is only possible if the schema matcher

is capable of linguistic analysis). Such subtypes as *different units* (e.g., prices in US$ vs. C$) would elude even the most advanced automatic matchers and require a human intervention.

- Other heterogeneity types (such as *implicit data* or *derivable data* [cf. Section 2.2.3]) are not encoded in either the schema or the instances of XML documents at all. Therefore no automatic schema matcher (no matter how sophisticated) can replace a domain expert in this case.

Even if we disregard all of the above concerns and apply automatic schema matching to the heterogeneities that it tackles best (specifically the *naming heterogeneity* [cf. Section 2.2.1] and *structural heterogeneity* [cf. Section 2.2.7] the result is still far from perfect. As *Wang at al.* say in [62]:

> *Existing automatic tools ... often require significant manual effort to correct incorrect matches and to add missing matches. In practice, schema matching is still done manually by domain experts, usually with a graphical tool [53, 49], and is very time consuming when ... schemas are large and/or complex.*

While in some circumstances such errors could be tolerated, this does not apply to healthcare (and many other domains), where a translation mistake might endanger the patient's life. However, this does not mean that the automatic schema matching should not be used at all. We merely contend that, unless significant progress is made, it should be used in a user-guided semi-automatic fashion.

In this thesis we do not concentrate on automating the schema matching process, however our approach could be easily extended in this direction, thus it is one of most likely future research areas [cf. Chapter 7].

4.2.2 Translation Specifications

As outlined in Section 4.1, originally the transformation programs were manually written in XSLT (following the identification of the semantic matches between the two heterogenous schemas). Because of the complexity of XSLT, this process was quite complicated and required intricate knowledge of XSLT itself. Many drastically different approaches designed to alleviate the effort and error-proneness of specifying the translation were proposed. As such, *translation specification* is less formalized and more diverse area than *schema matching*:

Visual XML editors with XSLT support: Since a graphical representation of XML documents and their schemas is easier to grasp than a textual one, many visual XML editors (e.g., XML Spy [26] and MarrowSoft <Xselerator> [20]) were created. Along with the visual representations (usually as a tree) of the XML files, these tools provide useful features (such as syntax highlighting, debugging and sometimes transformation templates) for the creation of XSLT programs. However, these documents are still specified in textual from only (i.e., as plain XSLT), requiring the user to master the language before he/she can specify any moderately complex translation.

XSLT Stylesheet editors: XSLT was originally designed as a part of XSL [cf. Section 2.1.2.4] with the goal of converting XML document into a presentation format (usually HTML or XHTML). A number of tools such as XSLerator [13] and <XSL>Composer [25] were developed for designing XSL stylesheets for this purpose. While some of these tools are very sophisticated, they are also limited to this particular application and are not suitable for general XML transformations.

Simple mappers: As discussed above, in order to figure out how to translate an XML document between two heterogenous schemas, we have to determine which element(s) of the source schema correspond to which element(s) of the target schema. Since humans are largely "visual creatures", the easiest way to do so is to draw the connections between corresponding elements on the graphical representations of the schemas. The earliest tools to use this approach, *simple mappers* such as VisualXSLT [7] were capable only of mapping a single source element to a single target elements (and without any additional transformations), thus, while very easy to use, they were quite limited in types of heterogeneity that they could resolve. i.e., they could handle only *naming heterogeneity* [cf. Section 2.2.1] and some of the simpler subtypes of the *structural heterogeneity* [cf. Section 2.2.7].

Operation-based mappers: Since, while being a considerable improvement on previous approaches from the usability perspective, *simple mappers* were inade-

quate for many translations, that approach was extended into *operation-based mappers*. Operation-based mappers (e.g., XSLWiz [14] and TagFree X2X [23]) can take a set of the elements from the source schema, apply an operation to them (e.g., concatenate the values) and map the result of this operation to an element of the target schema. This approach is considerably more powerful than *simple mapping* and can potentially, depending on what operations are supported be the tool, resolve most types of heterogeneity (provided, of course, that the user would detect the heterogeneities that have to be resolved). The only significant drawback of the *operation-based mappers* is that the result of an operation could be only mapped to the target schema and could not be used as an argument for another operation.

Multistage mappers: This last drawback was resolved by the introduction of *multistage mappers* that allow the output of any operation to be used as an input for another operation. *Multistage mapping* is the most powerful of all mapping-based translation specifications, however its power as well as usability varies between different implementations. For example, StylusStudio XML Mapping Tools [22] restricts the available operations to XSLT constructs (such as <value-of ...>) and functions, thereby requiring the user to know XSLT (which partially defeats the purpose of using visual abstraction instead of textual programming), whereas MapForce [19] and BizTalk Mapper [16] even allow the use of user-defined operations.

Alternative languages: In an completely orthogonal approach, various alternative transformation languages, designed to replace XSLT, were proposed by different research groups:

STX (Streaming Transformations for XML) [31]: Syntactically very similar to XSLT, however STX transformations are based on SAX events rather than on XPath matches. As such, it is much faster and more memory efficient than XSLT.

FXT (Functional XML Transformer) [29]: This language is also syntactically similar to XSLT. Authors claim that the advantages of FXT are mainly contained in its backend and pattern matcher:

> *The idea of fxt is to combine the computational power of SML, a functional programming language well suited for tree processing (which is to what XML document processing mainly reduces),and the expressivity of pattern matching over XML documents, provided by fxgrep, a powerful pattern matcher for XML documents.*

Declarative languages: A number of languages, such as [61] and [58], were designed as high-level specification languages, that could be used by non-programmers to specify document transformations descriptively. Such descriptions are then translated into XSLT programs.

VXT (Visual XML Transformer) [51]: VXT is an example of a visual transformation language. It was designed to provide visual support for data

structure representation and manipulation, so that the user *"will be freed from maintaining complex mental models of data structures, which play a central role in the transformation specification."* [51]. Transformations are specified visually, however, to the detriment to the power and flexibility of the VXT, they are limited to the ones defined by XSLT.

4.3 Transformation-based Approach

The mapping approach to the translation specification seems to be prevalent today, however, despite their power (and partly because of it) both single- and multistage operation-based mappers have several handicaps:

- Since the visual representations of both schemas plus all of the used operations and connections between them are displayed simultaneously, the screen becomes very cluttered and after a certain point it becomes hard to analyze the schemas and to continue creating a translation specification.
- For the same reason as above, it is also hard to judge when the specification is complete.
- And most importantly, due to the number of available operations, the ability to define new ones, flexibility of their use (ease of interconnections) and the lack of formal definitions, these mapping approaches are not well suited for reasoning about the translation.

Thus, we have decided to employ a transformation-based approach (a similar approach was outlined but not formally defined in [59]) in order to remedy these issues, while still being able to resolve as many heterogeneity types as possible. Our approach can be summarized as follows:

1. We propose a graph data model for DTDs [cf. Section 5.1] (the model can be extended to accommodate XML Schemas as well).
2. We have formally defined a set of transformation operations based on graph rewriting rules [cf. Section 5.4] along with the properties of such operations [cf. Section 5.3]. Each operation consists of two parts: schema transformation that modifies the DTD graph and an instance transformation that modifies the XML documents accordingly [cf. Figure 4.2].
3. In order to generate a translation specification, the user would iteratively apply the predefined transformation operations to the source schema graph. With each iteration, this graph should become more and more similar to the target schema graph. Once the graphs match, the transformation specification is completed and an XSLT program that can translate XML documents from source to the target schema is generated (based on the instance transformation parts of our operations). Note that since most (unfortunately it is not possible to do this automatically in *all* cases) of our operations specify both direct and inverse transformation, in many cases the translation from target to the source schema would be generated as well. The resulting transformation program(s) would

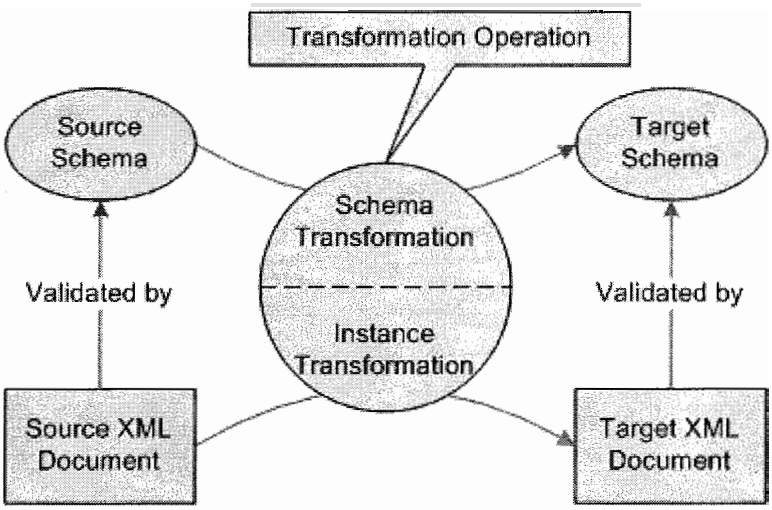


Figure 4.2. *Transformation process*

be used by the *Transducer* component [cf. Section 3.3.2.4] of our *HealthMatrix* network [cf. Chapter 3].

The most important advantage of our approach[3] is that instead of the jumble of operations and connections (resulting from a mapping approach), our translation specification is comprised of a simple sequence of transformation operations and each operation has well defined properties. This will allow us to reason about the information loss (and/or new information being introduced) during the translation. For example, it will enable us to ensure that the information that we would like to delete during the translation (e.g., for the anonymization in EMR) is really being deleted and that critical data is not accidentally modified.

In addition to that, from a usability perspective, our approach can simplify the comparison between the schemas, since at every iteration the user is presented with

[3]We will discuss the impact of our approach on resolving the heterogeneities in Section 6.2.

the new version of the source schema that is closer to the target schema than the previous version and is not distracted with extraneous items (such as icons for the operations and connections between them). This iterative process should also simplify automation of the schema matching (we do not currently use it, but will likely do so as future research).

An unfortunate drawback of our approach, however, is that it is much less intuitive and requires more planning than the simple connection between elements (and operations) that the mapping paradigm provides.

Chapter 5

Transformations

5.1 Representing the schema

The schema of an XML document is described in the DTD [17] as a list of element and attribute declarations. Every element has a name and can have a *content particle* and/or attributes or a *data particle*. *Data particle* can be *ANY* for an element containing arbitrary *content particles*, *EMPTY* for an empty element and *#PCDATA* for element containing text only. *Content particle* can be an element or can be comprised of other *content particles* grouped as a sequences *(a,b)* or as choices *(a|b)*. A number of occurrences of a *content particle* in its parent element is specified using quantifier "?" for optional, "+" for one-to-many and "*" for zero-to-many.

Attributes of an element have names and can have different types. The types are separated into three kinds: string (*CDATA*); a predefined type from XML specification [17] (*ID, IDREF, IDREFS, ENTITY, ENTITIES, NMTOKEN, NMTOKENS*); or an enumeration of possible values. Attributes could also be mandatory (*#REQUIRED*) or optional (*#IMPLIED*). If an attribute is neither mandatory nor optional, a default value can be provided. This value can also be declared constant

(*#FIXED*).

In order for us to apply the transformation operations (defined as graph rewriting rules), the DTD has to be modeled as a graph. Nodes of the DTD graph are divided into the following types [cf. Figure 5.1]:

SchemaElement: SchemaElement is an abstract type that represents any node in the DTD graph.

Content node: Content node is an equivalent of the DTD's *content particle*. In the graph, its instances are used to group other *content nodes* (including elements) together.

GroupType: Specifies how the children of this node are grouped together (as a sequence or as choice).

Quantifier: Specifies how many times this node can occur in its parent. "?" for optional, "+" for one-to-many and "*" for zero-to-many.

Element node: Each element node represents an element defined in the DTD and has the following properties:

Name: Name of this element.

GroupType: Same as the corresponding property of the *Content node*.

Quantifier: Same as the corresponding property of the *Content node*.

Attribute node: Each attribute node represents an attribute defined in the DTD and has the following properties:

Name: Name of this attribute.

AttributeType: Type of this attribute: *CDATA*, one of the predefined types (e.g., ID) or an enumeration.

DefaultType: Type of this attribute's default state: *#REQUIRED*, *#IMPLIED*, *#FIXED* or null.

DefaultValue: Attribute's default value (only if *DefaultType* is *#FIXED* or null).

Data: Data is an abstract type that represents a DTD *data particle*. Indicates the contents of its parent element. It has the following subtypes:

Any: The parent element contents is arbitrary.

Empty: The parent element is empty.

Text: The parent element only contains text.

A DTD graph can have only a single root node and this node has to be an element. Edges of the graph represent the parent/child relationship between nodes. *Content nodes* can have *content nodes* and *elements* as children. *Elements* can have *content nodes*, *elements*, *attributes* and *data nodes* as children.

An example of a DTD and its corresponding graph is shown in Figure 5.2.

5.2 Simplifying the schema

For any given DTD (or XML Schema) that defines a set of compliant XML documents, there exist infinitely many equivalent DTDs that define the same set of documents.

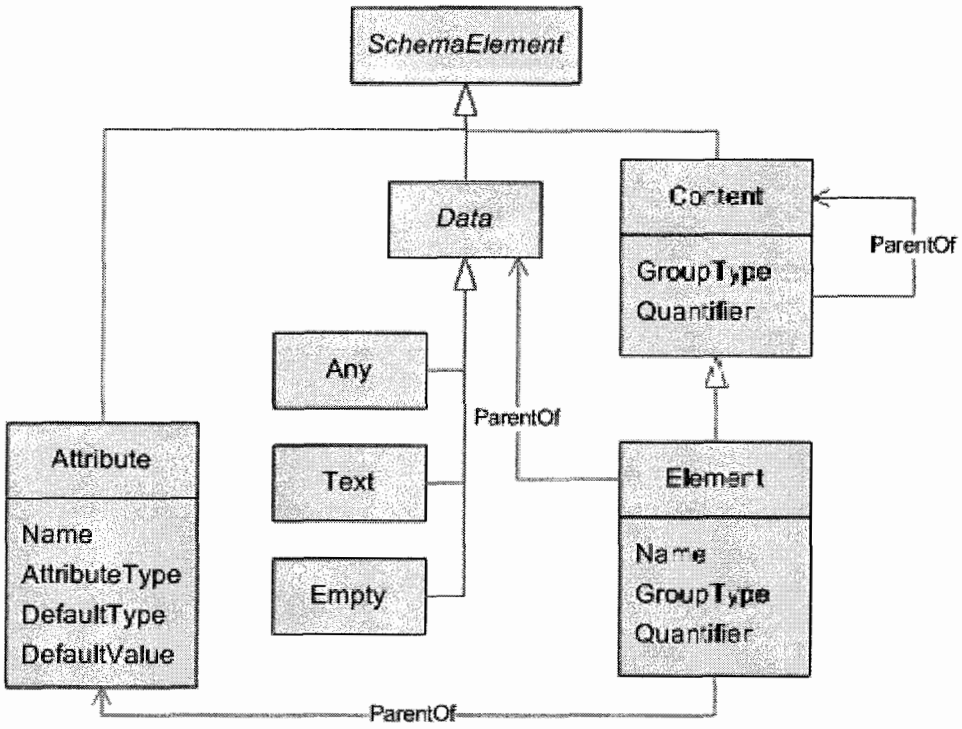


Figure 5.1. *Nodetypes of the DTD graphs*

These alternative DTDs could, for example, be derived from the original DTD by grouping the content particles differently, nesting such groupings, applying different quantifiers, etc. Such multiplicity of equivalent DTDs compounds the translation problem, since a pair of DTDs that look quite different, might be in fact very similar. To mitigate this problem we have defined a number of rules [cf. Figure 5.3] that can be used to simplify the schemas, while not affecting the documents conforming to them. For the sake of simplicity, we are expressing these rules using DTD formalism; the same rules could easily be expressed as graph rewriting productions in order to apply them to our schema graphs.

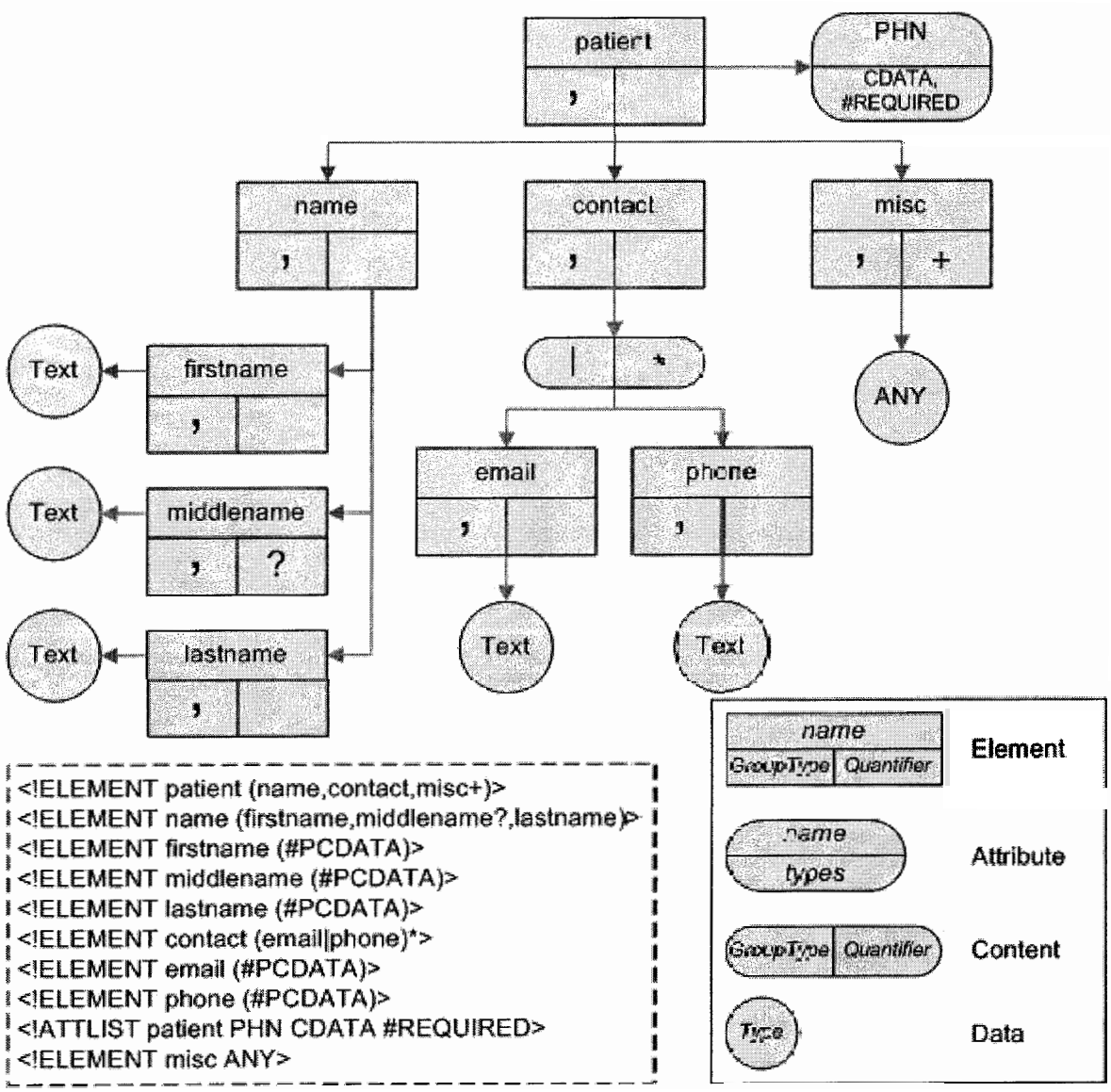


Figure 5.2. *Example of a DTD with its graph representation*

$((C)) \Leftrightarrow (C)$
$((C))\# \Leftrightarrow ((C)\#) \Leftrightarrow (C)\#$, *where # is any quantifier*
$((C)?)? \Leftrightarrow (C)?$
$((C)*)* \Leftrightarrow (C)*$
$((C)+)+ \Leftrightarrow (C)+$
$(\ldots,(b,c),\ldots) \Leftrightarrow (\ldots,b,c,\ldots)$
$((C)*)+ \Leftrightarrow ((C)*)? \Leftrightarrow ((C)+)* \Leftrightarrow ((C)+)? \Leftrightarrow ((C)?)* \Leftrightarrow ((C)?)+ \Leftrightarrow (C)*$

Figure 5.3. *Schema simplification rules*

5.3 Properties of the Transformations

5.3.1 Information Capacity Modification

One of the advantages of our translation approach is the ability to reason about the data loss (and/or the creation of new data) as a result of the XML document translation. To accomplish this, we classify individual transformations based on their effect on *information capacity* [43] of the schema. An easily understandable definition of *information capacity* is provided in [47]:

> *A schema S2 has more information capacity than schema S1 if every instance of S1 can be mapped to an instance of S2 without loss of information. Specifically, it must be possible to recover the original instance from its image under the mapping.*

Thus, our transformations fall into the following categories:

Information Capacity (IC) Increasing: Transformations that result in new information being added (e.g., adding a new attribute to an element) are consid-

ered IC Increasing.

IC Reducing: Transformations that result in information being lost (e.g., deleting an element) are considered IC Reducing.

IC Preserving: Transformations that neither reduce nor increase the amount of information (e.g., renaming an element) are considered IC Preserving.

IC Ambiguous: For some operations, however, the loss, preservation or increase of information could not be derived from the schema alone and depends on an instance of the XML document being translated. Let's consider, for example, the change of an element's quantifier from '*' to '' (*required*). If in the original document the corresponding element occurred 0 times, then, in order to satisfy the new constraint we will have to add an instance of this element to the target document and the operation is *IC Increasing.* If this element occurs exactly once, then we don't have to do anything and the operation is *IC Preserving.* However, if the corresponding element occurs more than one times in the source document, the transformation would have to remove all occurrences but one and the operation is *IC Reducing.* Such transformations are considered IC Ambiguous.

5.3.2 Reversibility

The transformation is *reversible* if, based on its input parameters, we can derive the parameters for its inverse transformation. For example, if the forward transformation

is 'delete element' then the inverse operation would simply create such an element at its former position. The purpose of defining an inverse for each transformation is to enable us to generate translators between two schemas in *both* directions without any additional efforts. Thus, this does not necessarily mean that if we apply a forward and than an inverse transformation to a particular XML document we will get the original document as a result; all that it ensures is that the resulting document will conform to the original schema again. That is, the inverse transformation will recover the schema, but will not recover the instance if one of the operations (forward or inverse) is *IC Reducing* or if an operation on an element's content is not reversible (e.g., if the original transformation computed the element's value by multiplying the values of several other elements, it is impossible to compute the values of the source elements based just on the result of this multiplication).

5.4 Transformations

Definition of each transformation includes the following elements:

Description: Outlines the purpose of the transformation.

Parameters: List of the input parameters for the transformation.

Preconditions: Conditions that have to be true in order for the transformation to be applicable

Schema Transformation: Visual representation of the transformation as the graph rewriting rule for the schema graph. Includes the representation of precondi-

tions as well. The graph rewriting notation used in this thesis is explained in Section 2.3.

Instance Transformation: XSLT program that performs this transformation on the XML documents. An example of the instance transformation is shown for the *Delete Element* operation [cf. Section 5.4.1].

Information Capacity Modification: Shows how the transformation affects *information capacity.*

Inverse Transformation: Specifies the corresponding inverse transformation (along with its parameters).

Result: Description of the effect that the transformation has on both the schema and XML documents.

5.4.1 Delete Element

Description: Deletes an element.

Parameters:

E: element to be deleted.

path: path from the schema root to the element's parent (to uniquely identify E).

Preconditions: Element E has to be *empty* or contain only *text. Note: in order to delete an element that has other types of children, these child elements have to be deleted first.*

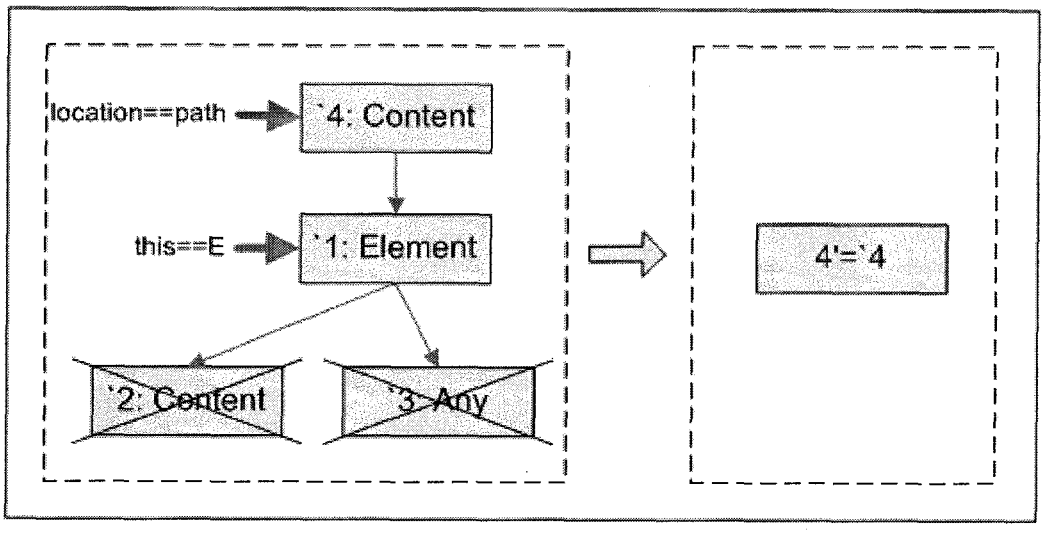


Figure 5.4. *Delete Element*

Schema Transformation: See Figure 5.4.

Instance Transformation: $*ElementPath*$ is an XPath expression for ***path*** that points to an element to be deleted.

```
<xsl:stylesheet>
  <!-- Copy document -->
  <xsl:template match="node()|@*">
    <xsl:copy>
     <xsl:apply-templates select="@*|node()"/>
    </xsl:copy>
  </xsl:template>
  <!-- Delete element -->
  <xsl:template match=$ElementPath$/>
```

```
</xsl:stylesheet>
```

Information Capacity Modification: This transformation is *IC Reducing.*

Inverse Transformation: *Create Element* with $E_{create} = E; path_{create} = path;$ $defaultValue_{create} = EMPTY$.

Result: The element E along with its attributes and *Text* or *Empty* data nodes (if they exist) is removed from the schema. All instances of this element are deleted from the translated XML document.

5.4.2 Create Element

Description: Creates a new element.

Parameters:

E: element to be created.

path: path from the root of schema to the location of the new element's parent.

defaultValue: default value for this element.

Preconditions: No elements with the same name ($E.name$) should already exist.

Schema Transformation: See Figure 5.5.

Information Capacity Modification: This transformation is *IC Increasing.*

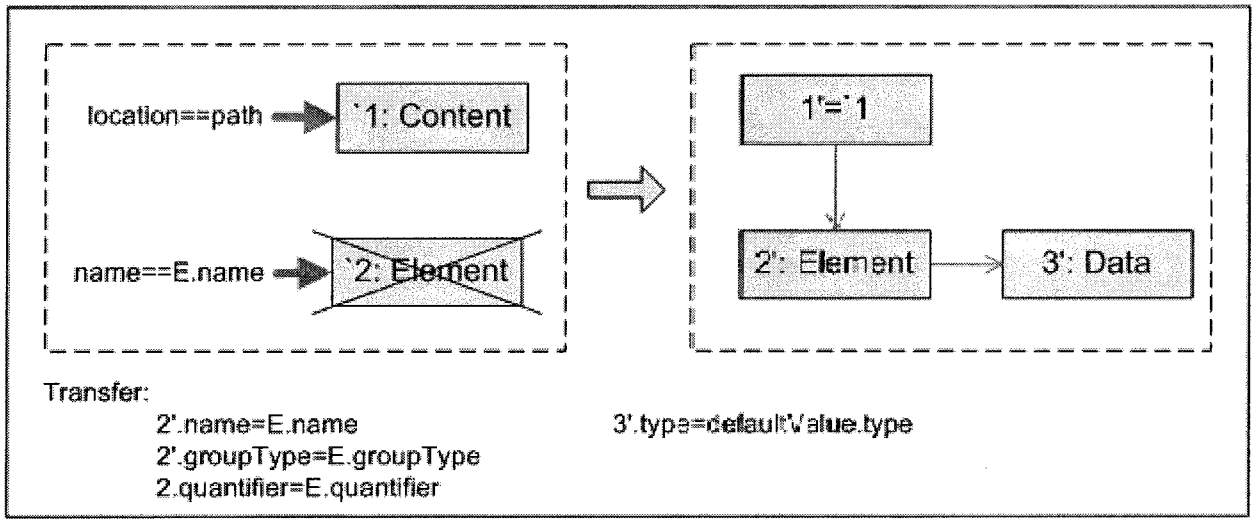


Figure 5.5. *Create Element*

Inverse Transformation: *Delete Element* with $E_{delete} = E; path_{delete} = path$.

Result: In the schema, the element E is added at the location pointed to by *path*. Depending on the type of *defaultValue*, either an *Empty* or a *Text* node is created as the child of E. In the translated XML document, an instance of this element with the corresponding value is created at each location that matches *path*. *Note that only one instance of the element is created, since it would be correct regardless of the* E's *quantifier.*

5.4.3 Delete Attribute

Description: Deletes an attribute.

Parameters:

A: attribute to be deleted.

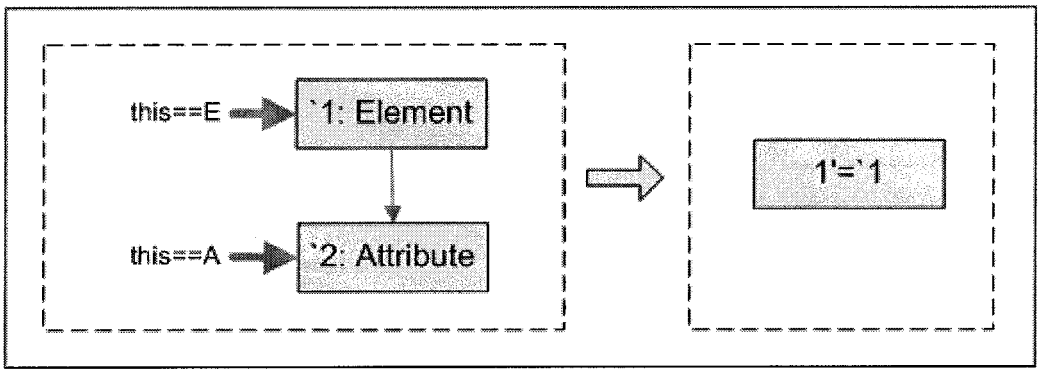


Figure 5.6. *Delete Attribute*

E: element that contains this attribute.

Preconditions: None

Schema Transformation: See Figure 5.6.

Information Capacity Modification: This transformation is *IC Reducing.*

Inverse Transformation: *Delete Attribute* with $E_{create} = E; A_{create} = A$. If $A.defaultType$=$\#FIXED$ then $value_{create} = A.defaultValue$, otherwise $value_{create} =$ "".

Result: The attribute A definition for the element E is removed from the schema. All instances of this attribute are deleted from the instances of the element E in the translated XML documents.

5.4.4 Create Attribute

Description: Creates a new attribute.

Parameters:

A: attribute to be created.

E: parent element of this attribute.

value: value for this attribute

Preconditions:

1. No attribute with the same name ($A.name$) is already defined for element E.
2. If $A.defaultType$ is $\#FIXED$, then $value$ has to be the same as $A.defaultValue$.

Schema Transformation: See Figure 5.7.

Information Capacity Modification: This transformation is *IC Increasing.*

Inverse Transformation: *Delete Attribute* with $E_{delete} = E; A_{delete} = A$.

Result: In the schema, the attribute A is added at the element E. In the translated XML document, an instance of this attribute is created at each instance of the element E. If $value$ is specified (and obeys precondition 2) then attribute's value is set to $value$, otherwise it is set to $A.defaultValue$.

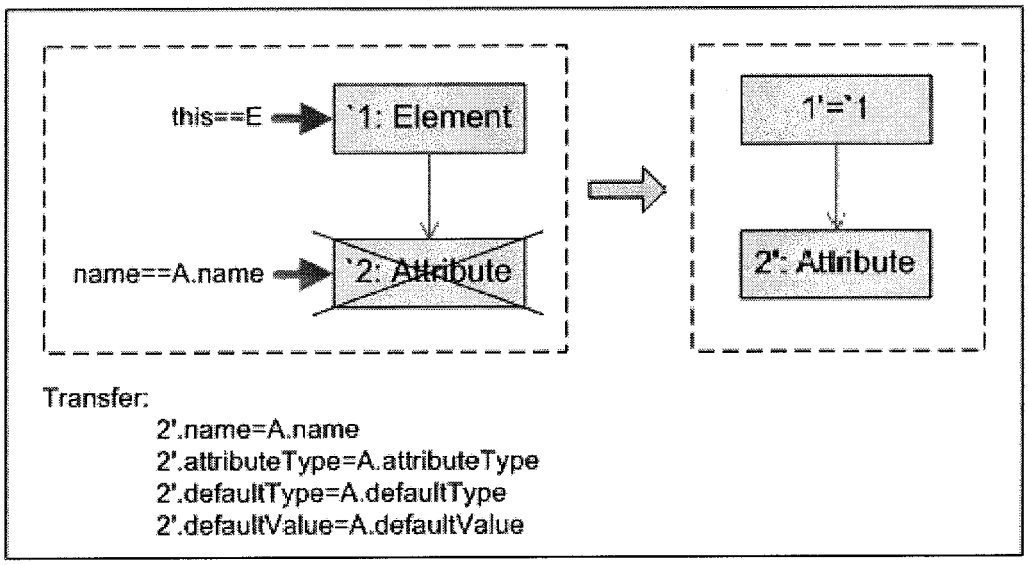


Figure 5.7. *Create Attribute*

5.4.5 Convert Attribute to Element

Description: Converts an element's attribute into its subelement.

Parameters:

A: attribute to be converted.

E: parent element of this attribute.

Preconditions: No element with the same name as the converted attribute (*A.name*) is already defined for the schema.

Schema Transformation: See Figure 5.8.

Information Capacity Modification: This transformation is *IC Preserving.*

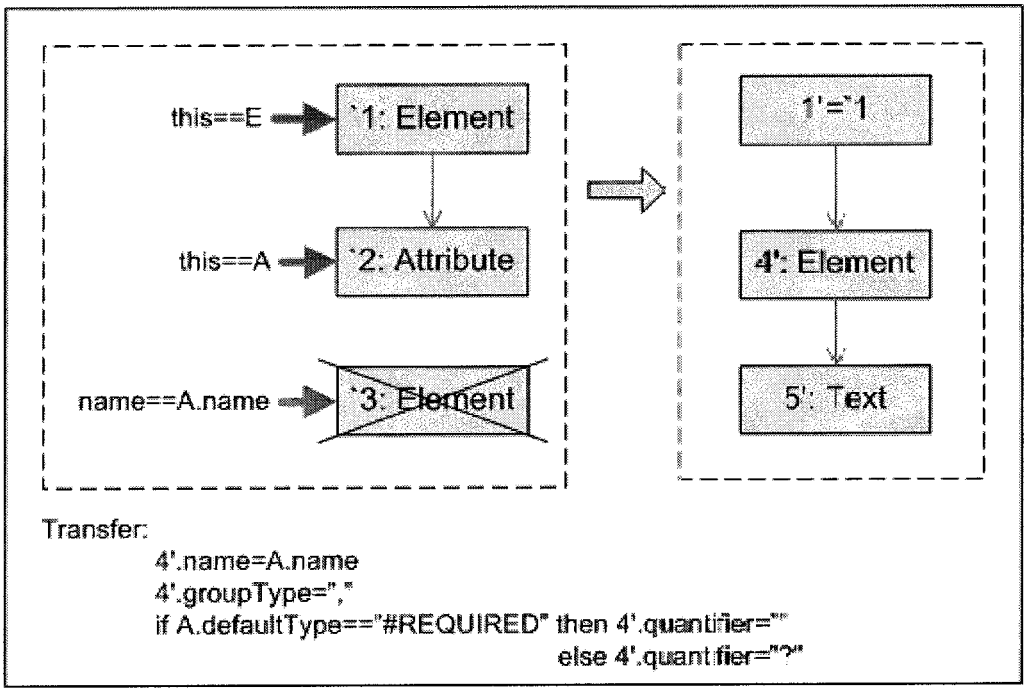


Figure 5.8. *Convert Attribute to Element*

Inverse Transformation: *Convert Element to Attribute* with $P_{e\_to\_a} = E; E_{e\_to\_a} = A$.

Result: In the schema, an attribute node A is removed from the element E and a new subelement equivalent to A is added along with its *text* node child. The *groupType* of the new element is always *sequence*. If the attribute is optional (*defaultType* is not *#REQUIRED*) then the new element's *quantifier* is set to"?".

In the translated XML document, each instance of this attribute is replaced with an element with the same name and value.

5.4.6 Convert Element to Attribute

Description: Converts an element into an attribute of its parent element.

Parameters:

E: element to be converted.

P: parent element of this element.

Preconditions:

1. No attribute with the same name as the converted element (*E.name*) is already defined for element *P*.
2. Element *E* has to be *empty* or contain only *text*.
3. *E's* quantifier has to be *none* or "?".

Schema Transformation: See Figure 5.9.

Information Capacity Modification: This transformation is *IC Preserving*.

Inverse Transformation: *Convert Attribute to Element* with $E_{a\_to\_e} = P; A_{a\_to\_e} = E$.

Result: In the schema, an element node *E* along with its *text* (or *empty*) node child is removed from the element *P's* children and a new attribute equivalent to *E* is added. The *attributeType* of the new element is always *CDATA*. If the element's

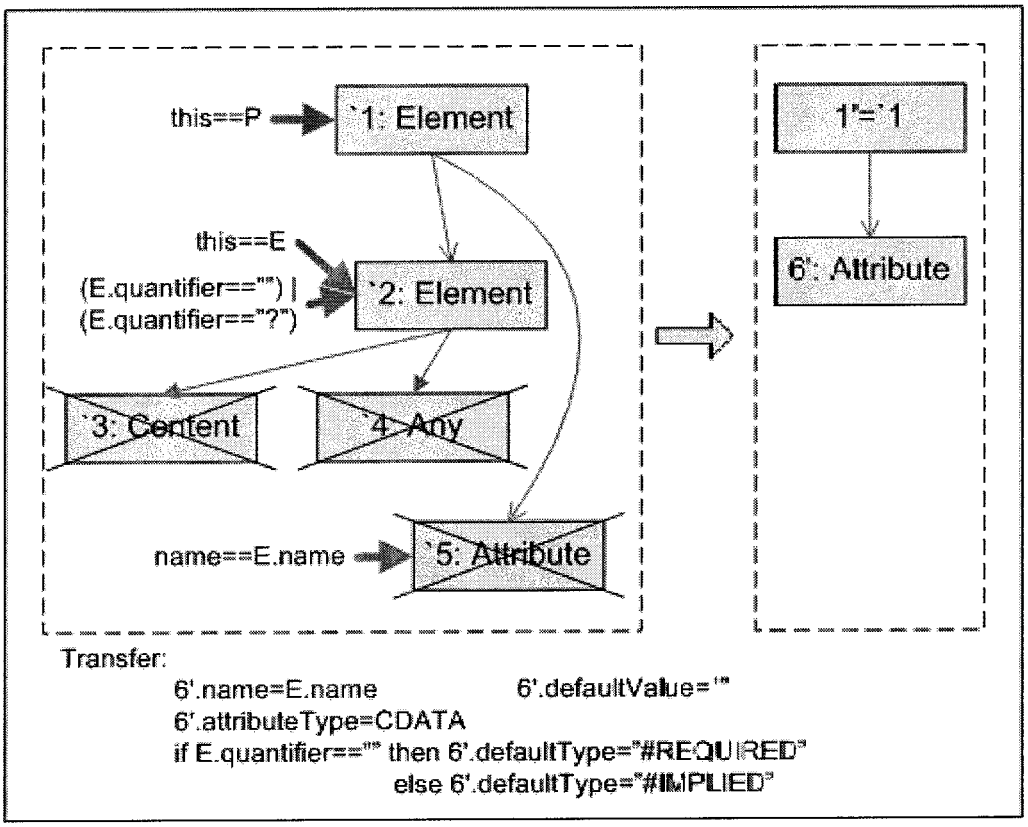


Figure 5.9. *Convert Element to Attribute*

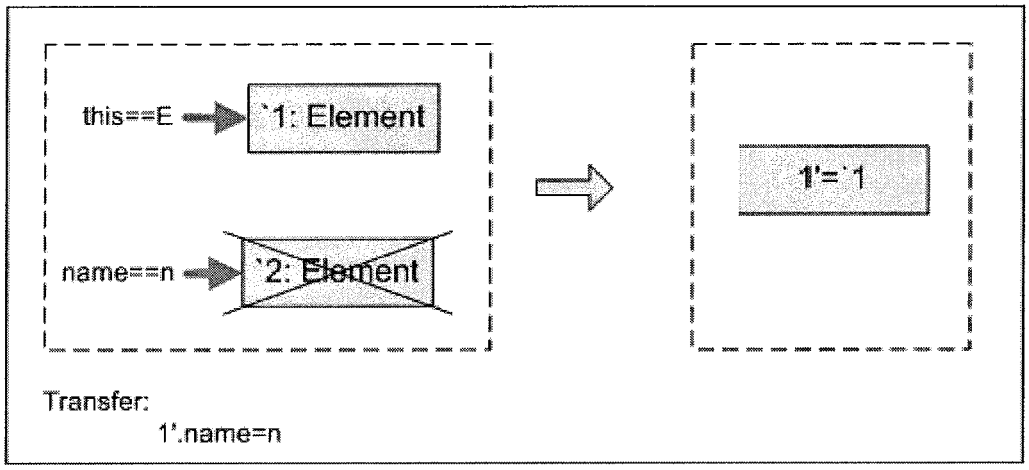


Figure 5.10. *Rename Element*

quantifier was *none* then the new attribute's *defaultType* is set to *#REQUIRED*, otherwise it becomes *#IMPLIED*.

In the translated XML document, each instance of this element is replaced with an attribute with the same name and value.

5.4.7 Rename Element

Description: Changes the name of an element.

Parameters:

E: element to be renamed.

n: new name of this element

Preconditions: No elements with the same name (n) should already exist.

Schema Transformation: See Figure 5.10.

Information Capacity Modification: This transformation is *IC Preserving.*

Inverse Transformation: *Rename Element* with $n_{rename} = E.name; E_{rename} =$ *E(with name=n).*

Result: In the schema, the name of the element is changed to a new one. In the translated XML document, each instance of this element is renamed to the new name.

5.4.8 Rename Attribute

Description: Changes the name of an attribute.

Parameters:

A: attribute to be renamed.

E: the parent element of this attribute.

n: new name of this attribute.

Preconditions: No attributes with the same name (n) should be already defined for this element (E).

Schema Transformation: See Figure 5.11.

Information Capacity Modification: This transformation is *IC Preserving.*

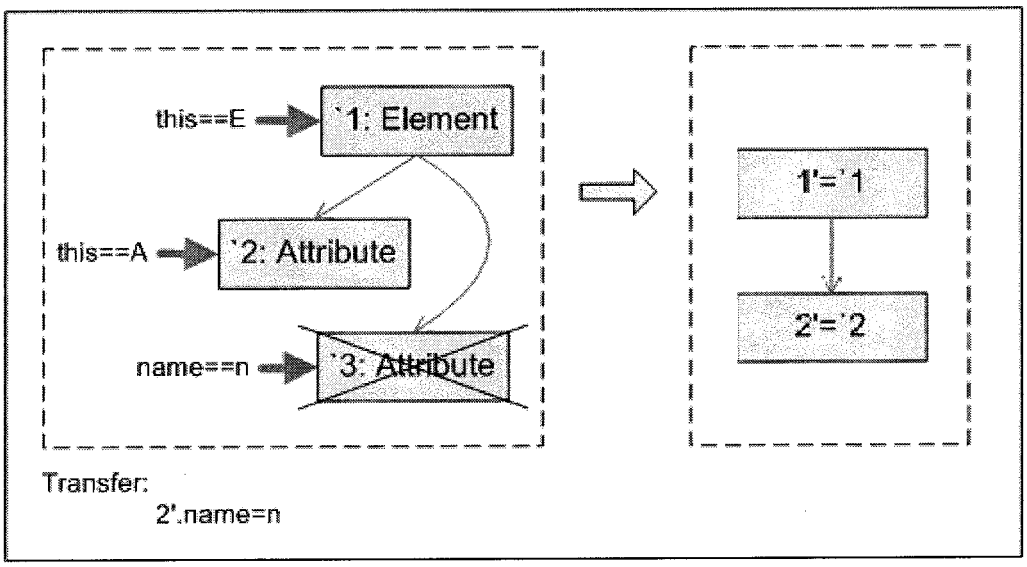


Figure 5.11. *Rename Attribute*

Inverse Transformation: *Rename Attribute* with $E_{remane} = E$; $n_{rename} = A.name$; $A_{rename} = A(with\ name{=}n)$.

Result: In the schema, the name of the attribute is changed to a new one. In the translated XML document, each instance of this attribute is renamed to the new name.

5.4.9 Move Content Node

Description: Moves the content node (and a tree rooted in it) to another location.

Parameters:

N: node to be moved.

oldPath: path to the current parent of this node.

newPath: path to the new parent of this node.

Preconditions: None.

Schema Transformation: See Figure 5.12.

Information Capacity Modification: This transformation is *IC Preserving.*

Inverse Transformation: *Move Content Node* with $N_{move} = N; oldPath_{move} = newPath; newPath_{move} = oldPathE.$

Result: In the schema, the *content node* (along with a subtree rooted in it) is moved from the old location to the new one. If, after this move, the old parent of the moved node doesn't have any remaining children, then an *empty* node is added to it. The corresponding change is performed in the translated XML documents.

5.4.10 Move Attribute

Description: Moves an attribute from one element to another.

Parameters:

A: attribute to be moved.

oldP: old parent element of this attribute.

newP: new parent element of this attribute.

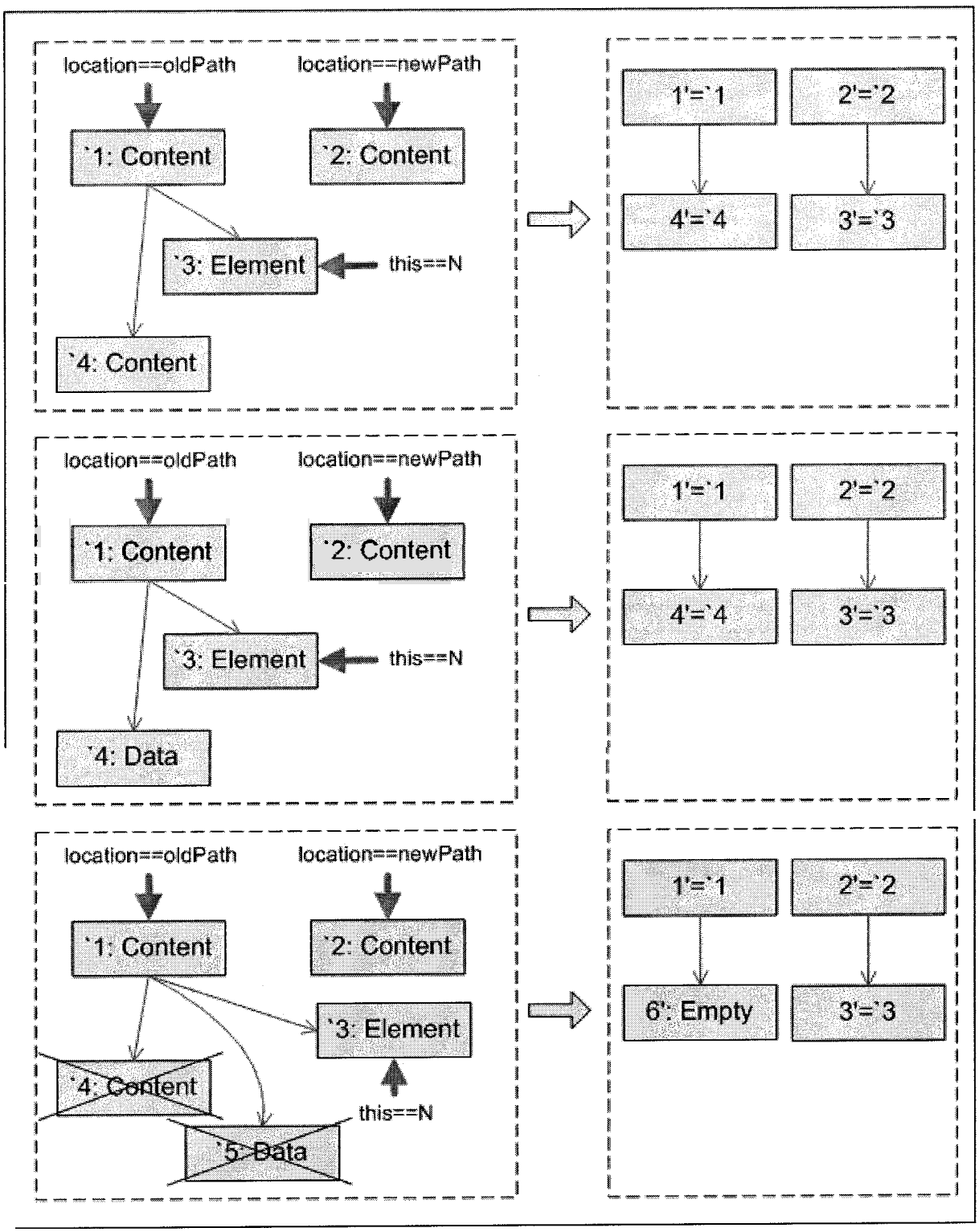


Figure 5.12. *Move Content Node*

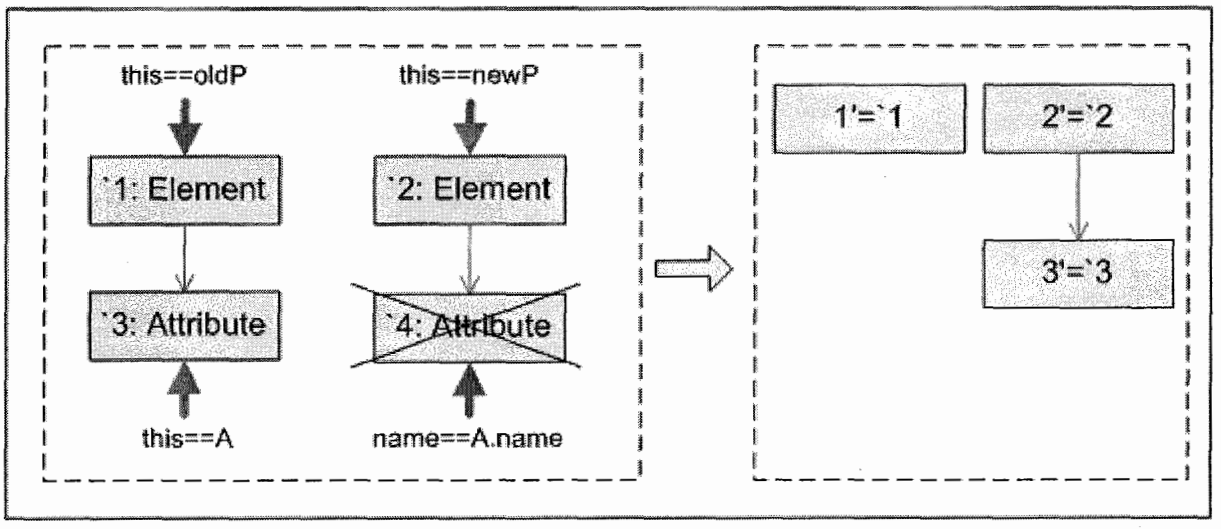


Figure 5.13. *Move Attribute*

Preconditions: No attributes with the same name ($A.name$) should be already defined for the new parent element ($newP$).

Schema Transformation: See Figure 5.13.

Information Capacity Modification: This transformation is *IC Preserving.*

Inverse Transformation: *Move Attribute* with $A_{move} = A$; $oldP_{move} = newP$; $newP_{move} = oldP$.

Result: In the schema, the attribute A is moved from the old element to the new one. The corresponding change is performed in the translated XML documents.

5.4.11 Convert to Group

Description: Groups a set of *content nodes* together under another *content node.*

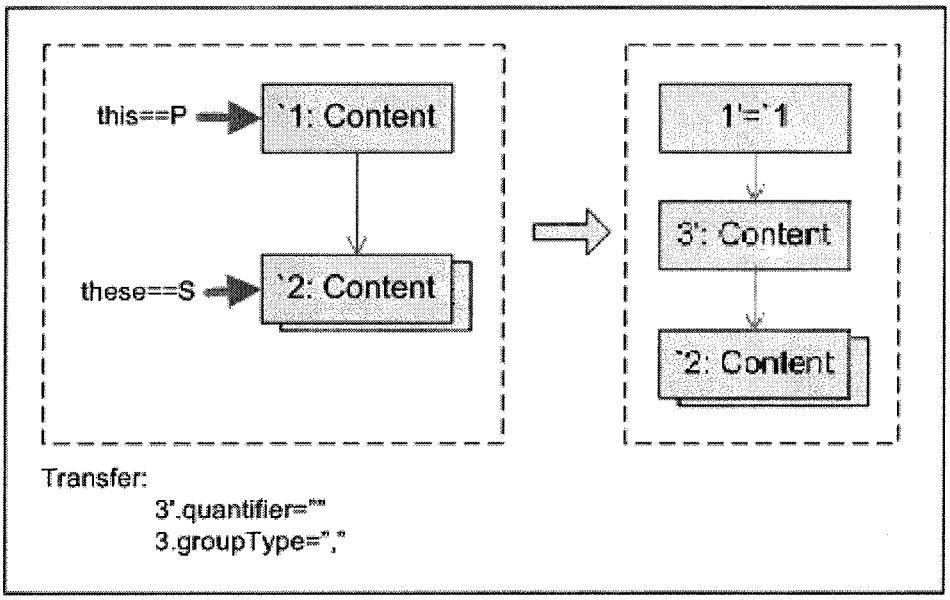


Figure 5.14. *Convert to group*

Parameters:

S: a set of content nodes to be grouped.

P: the current parent of these nodes.

Preconditions: All the nodes to group share a common parent (P)

Schema Transformation: See Figure 5.14.

Instance Transformation: None.

Information Capacity Modification: This transformation is *IC Preserving.*

Inverse Transformation: *Flatten Group* where $G_{flatten}$ is a new content node created by this operation.

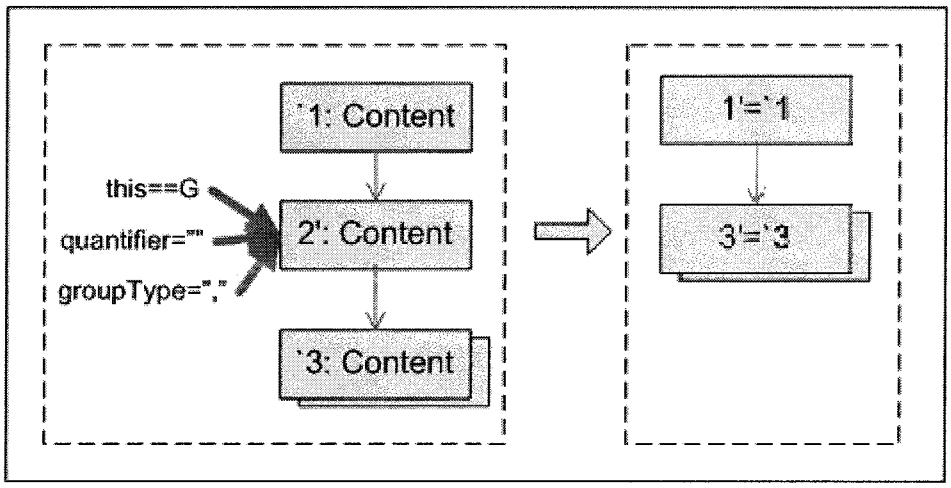


Figure 5.15. *Flatten group*

Result: In the schema, a new *content node* (with *quantifier*="" and *groupType*=",") is created as a child of P. All nodes from S are moved to be the children of this new node. This transformation does not modify the translated XML document.

5.4.12 Flatten Group

Description: Flattens the group of nodes (grouped under a content node).

Parameters:

G: a parent content node of the group to be flattened.

Preconditions: The parent content node (G) has to have *quantifier*="" and *groupType*=",".

Schema Transformation: See Figure 5.15.

Instance Transformation: None.

Information Capacity Modification: This transformation is *IC Preserving.*

Inverse Transformation: *Convert to Group* where S_{group} is the set of G's children and P_{group} is G's parent.

Result: In the schema, a *content node* G is deleted. All nodes that were the children of G become the children of its parent node. This transformation does not modify the translated XML document.

5.4.13 Change Attribute

Description: Changes the type and default parameters of an attribute.

Parameters:

A: attribute to be changed.

newAT: new attributeType.

newDT: new defaultType.

newDV: new default value.

Preconditions: If new defaultType is *#REQUIRED* or *#FIXED*, *newDV* must not be *null.*

Schema Transformation: See Figure 5.16.

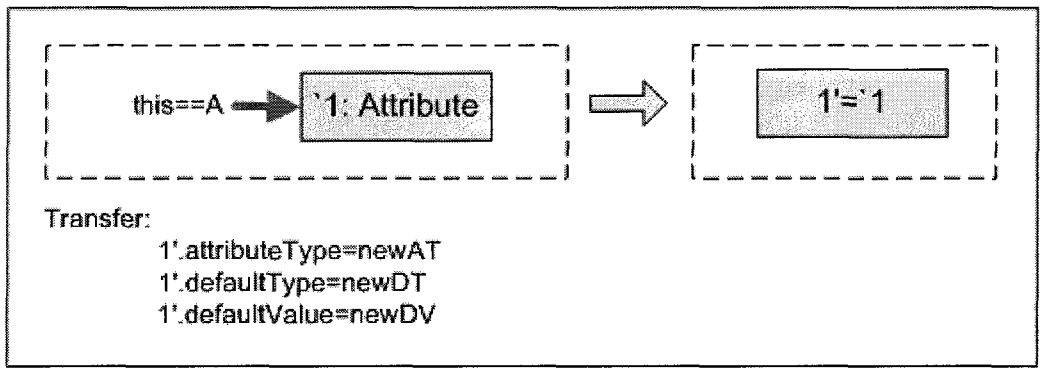


Figure 5.16. *Change Attribute*

Information Capacity Modification: The effect of this transformation on information capacity depends on the *defaultType* of the changed attribute. If the change is from any other type to *#REQUIRED*, then the transformation is *IC Reducing.* If the change is from *#REQUIRED* to any other type, then the transformation is *IC Increasing.* In all other cases the transformation is *IC Preserving.*

Inverse Transformation: *Change Attribute* with the original parameters of this attribute.

Result: In the schema, the parameters of the attribute A are updated.

The changes in the translated XML document depends on the old and new values of the parameters:

1. If the attribute's *default type* changed from *#IMPLIED* to *#REQUIRED*, the *newDV* has to be assigned to all instances of the attribute that didn't have a value before.

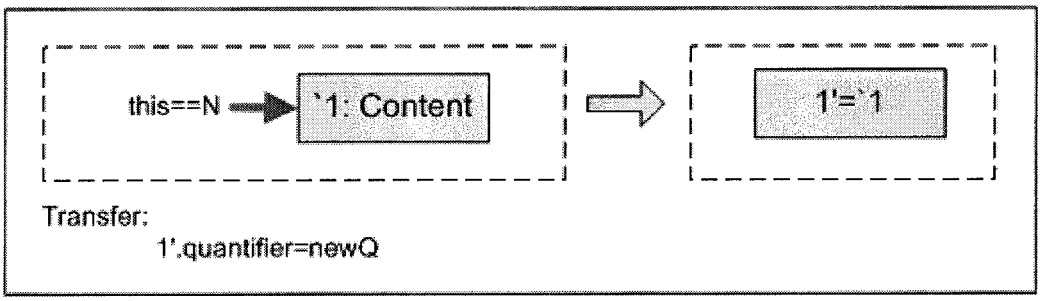


Figure 5.17. *Change Quantifier*

2. If the attribute's new *default type* is *#FIXED*, then the attribute's values has to be replaced with *newDV* for all instances of the attribute.

3. If the *default value* has changed, then for all instances of the attribute that have the value equal to the old default value, this value has to be replaced with *newDV*.

5.4.14 Change Quantifier

Description: Changes the quantifier of a *content node*.

Parameters:

N: content node to be changed.

newQ: new quantifier.

Preconditions: None.

Schema Transformation: See Figure 5.17.

From/To	?	+	*	.
?	Preserving	Ambiguous	Increasing	Reducing
+	Ambiguous	Preserving	Increasing	Reducing
*	Reducing	Reducing	Preserving	Ambiguous
.	Increasing	Increasing	Ambiguous	Preserving

Table 5.1. *Effect of* Change Quantifier *on information capacity*

Information Capacity Modification: The effect of this transformation on information capacity depends on the parameters of the old and new quantifiers and is explained in more details in Table 5.1 ('.' means *required*; original quantifiers are in first column; new quantifiers are in top row).

Inverse Transformation: *Change Quantifier* with the old quantifier of this node.

Result: In the schema, the quantifier of the *content node N* is changed.

The changes in the translated XML document depends on the old and new values of the quantifier:

1. If the original quantifier was "?" and the new one is "+" or *required*, that means that the element has to occur at least once. Thus, if the element doesn't occur, it has to be inserted into the document (with *empty* as a value).
2. If the original quantifier was "+" and the new one is "?" or *required*, that means that the element has to occur at most once. Thus only the first occurrence is kept in the document and the subsequent occurrences are deleted.
3. If the original quantifier was "*" and the new one is *required*, that means that

the element has to occur exactly once. Thus, if the element doesn't occur, it has to be inserted into the document (with *empty* as a value). If the element occurs multiple times, only the first occurrence is kept in the document and the subsequent occurrences are deleted.

4. If the original quantifier was "*" and the new one is "?", that means that the element has to occur at most once. Thus only the first occurrence is kept in the document and the subsequent occurrences are deleted. If the element does not occur at all, no action is needed.

5. If the original quantifier was "*" and the new one is "+", that means that the element has to occur at least once. Thus, if the element doesn't occur, it has to be inserted into the document (with *empty* as a value). If the element occurs multiple times, no action is needed.

5.4.15 Fold Over ID/IDREF

Description: In many cases (especially when the XML schema has been created based on the relational database schema) parts of the schema are linked via the use of *ID* and *IDREF* attributes. However, in some schemas, the same link might be expressed by making one such part a child of another. This difference can be resolved by folding the linked schema, which is accomplished in this operation by copying the content of a referenced element into the referencing element.

Parameters:

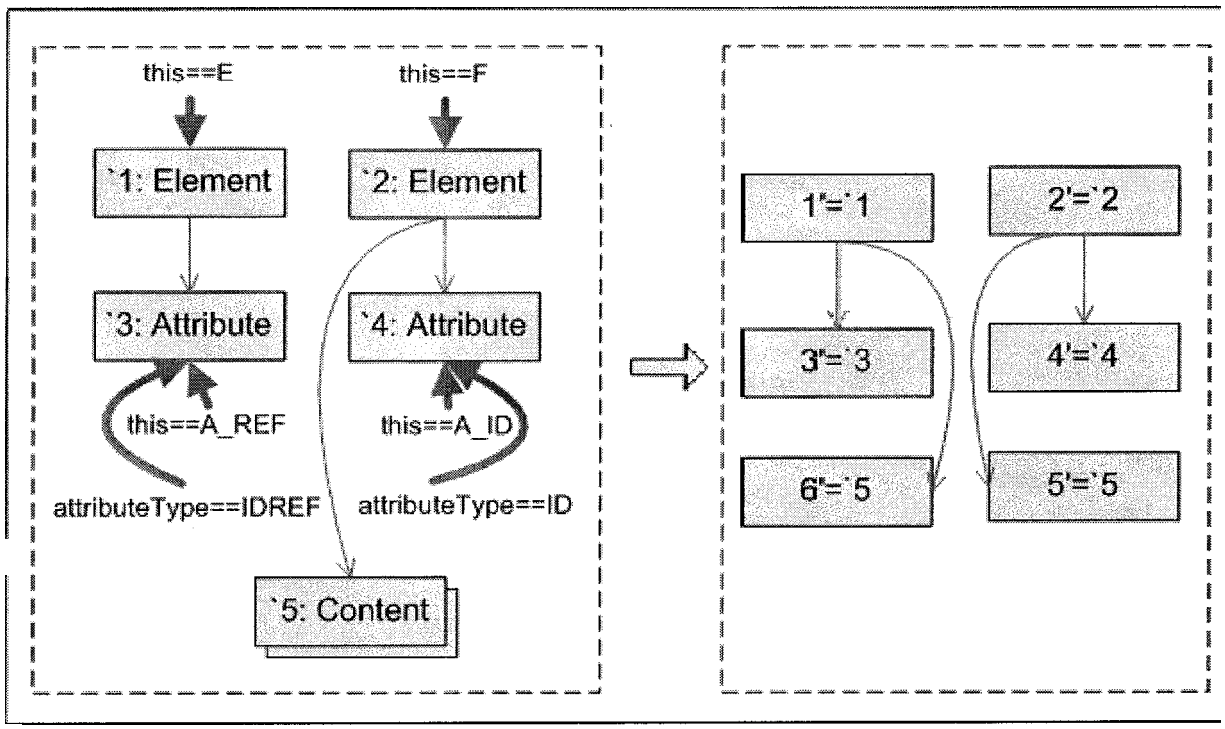


Figure 5.18. *Fold over ID/IDREF*

F: element with ID attribute.

A_ID: ID attribute.

E: element with IDREF attribute.

A_REF: IDREF attribute.

Preconditions:

1. *A_ID's attributeType* is *ID*.

2. *A_REF's attributeType* is *IDREF*.

Schema Transformation: See Figure 5.18.

Information Capacity Modification: This transformation is *IC Preserving.*

Inverse Transformation: *Delete Element* applied to the copy of *F's* content.

Result: In the schema, all child nodes of the F are copied to be the children of the E. In the translated XML document, the content of each element of type F is copied into the content of an element of type E with *A_REF* equal to *F's A_ID.*

5.4.16 Change Value

This transformation differs from the others by the fact that it does not affect the schema at all, but rather changes the values of the instances of attributes and elements in the translated XML files. *Change value* is a *meta-transformation*, since it takes a set of values of elements and attributes, uses these values as input parameters for an operation and assigns the result of this operation to be a value of another element or attribute. *Change value* can use a variety of operation to compute the result value and we are not going to list all of them in this thesis (also, eventually, the users should even be able to create custom operations, provided that they are formally defined). However, the following variants of *change value* could be used to resolve some of heterogeneity types defined in Section 2.2:

Copy: This is a very simple operation. It takes a value of an element or attribute and assigns it to another element or attribute. In conjunction with *create element/attribute* (that create an empty element/attribute) it can be used to make

a copy of an element. The inverse of this transformation simply copies the data in another direction.

Convert: By applying mathematical operations to the element/attribute values, *change value* can be used to convert them between different units (e.g., pounds to kilograms). This transformation is very useful in resolving numeric subtypes of *value heterogeneity* [cf. Section 2.2.2]. The inverse of this transformation is not always possible and depends on the employed mathematical operations and the way they were used (e.g., the pound/kg conversion can easily be reversed, while the multiplication of two element values can not).

Concatenate/Split: This variant of *change value* can concatenate the values of several elements/attributes (possibly interleaving them with ' ' or ','), thus resolving a very common *single element/multiple elements* subtype of *structural heterogeneity* [cf. Section 2.2.7]. Its inverse, *split*, separates a part of an element/attribute value (if such separation is possible, e.g., using ' ' or ',' as delimiters).

Mapping: This variant of *change value* maps one set of values (e.g., a range of numbers) to another (e.g., {Very Low, Low, Normal, High, Very High}. It can be used to resolve *semantic data granularity* [cf. Section 2.2.4]. The inverse of this operation is possible, however, in many cases the new mapping would have to be specified .

5.5 Supporting the user

While the preceding sections express the core of our approach, some additions to it, which are described in this section, make it more efficient and make the process of translation generation and schema exploration easier and more intuitive for the user.

5.5.1 Mappings

As we have stated in Section 4.3, a considerable drawback of our transformation-based approach is that, unlike the mapping-based approaches [cf. Section 4.2.2], it doesn't provide the user with an intuitive way of specifying matching elements in the source and target schemas. To mitigate this problem, we can simulate the mapping approach for the user (of course, the user still should be able to use individual transformations as well), while internally representing such a mapping as a sequence of transformations.

A simple mapping between two leaf elements can be simulated by first renaming the source element to match the target element's name and then moving the renamed element to the corresponding location. Both of these operations are *IC Preserving*, so the mapping would be *IC Preserving* as well.

If we would like to map a leaf element to an attribute (or an attribute to an element), the previous transformation sequence has to be augmented with a *convert element to attribute* (or *attribute to element*) transformation. The resulting sequence would be *IC Preserving* as well.

To map a non-leaf element (along with the subtree rooted in it) to another non-leaf

element, we would have to obey the precondition that their corresponding subtrees have to be isomorphic (if the subtrees are not isomorphic originally, they can be first made so by the user, using other transformations). First we would rename all the corresponding elements and attributes (both the root and in the subtree) and then we can move the tree to the new location. Since it is composed only of renames and a move, this mapping is also *IC Preserving.*

5.5.2 Highlighting Dependencies

In order to facilitate the analysis of the schemas and the discovery of the matching elements between them, it is advantageous to be able to show the user which elements of the original source schema have influenced the elements of the current version of this schema.

Our transformations are formally defined as a graph rewriting rules. The left-hand side of each of these rules represents the input or the corresponding transformation, while the right-hand side represents the output. Since the translation specification is comprised of a sequence of individual transformations, it is possible to build a graph structure (based on this sequence and the imput/output dependencies of each transformation) that would capture the history of the dependencies between schema elements. The exact approach on how to construct such a *history graph* is outlined in [44].

Thus, using such a *history graph,* we can trace the dependencies between ele-

ments backwards and deduce which schema elements (and/or transformations) have influenced any element of the current schema.

5.5.3 Optimization

Since our translation approach relies on the user to select the proper transformation at each step, the resulting translation specification (which stores *all* applied transformations in sequence) might not be the most efficient. This can happen for a variety of reasons ranging from the user simply not seeing the best translation approach to applying inverse transformations (instead of *undo*) to correct his or her mistakes.

We can remedy this issue somewhat by optimizing the translation specification after it is completed. Undoubtedly, many such optimizations exist, but discovering them is a subject for future research [cf. Section 7.3]. However, below we outline two simple optimizations, that should improve the performance of our transformation approach.

For both of these optimizations, we assume the following:

1. T_n and T_m are two transformations in the translation specification (in the n^{th} and m^{th} position respectively) and $m > n$.
2. $In(T)$ is a set of input schema elements and $Out(T)$ is a set of output schema elements for operation T.
3. There is no operation T_k, where $k \in (n, m)$, such that $\exists e \in Out(T_n), e \in In(T_k)$ or $\exists e \in In(T_m), e \in Out(T_k)$ (i.e., we can change T_n and T_m without affecting

any transformations in between).

The two optimizations that we can perform then are:

Combine partial transformations: This optimization applies if the user performed some transformation in several steps, when it is possible to do it in one. For example, the element with the name a was first renamed to b and then to c, instead of renaming it directly to c. In this case, T_n and T_m have the same transformation type (one of {*rename element/attribute, move content node/attribute, change attribute/quantifier, change value*}) and $Out(T_n) = In(T_m)$. To optimize, we would remove T_m and replace T_n with $T(In(T_n), Out(T_m))$.

Remove (unintentional) undos: This optimization applies if the later transformation undos the earlier one. For example, the element with the name a was first renamed to b and then renamed back to a. In this case, T_m would be an inverse operation of T_n [cf. Section 5.4 for the specifications] and $Out(T_m) = In(T_n)$. To optimize, we would remove both T_n and T_m.

Chapter 6

Implementation and Evaluation

6.1 Implementation

In order to test our transformation-based approach, we have created a prototype implementation. The prototype consists of two parts: a tool (called ***Odin***[1]) that allows the user to define a translation specification and a *transducer* component [cf. Section 3.3.2.4] for our *HealthMatrix* network that translates the information units between different schemas.

6.1.1 Odin

The Odin prototype is somewhat limited, specifically in the GUI for specifying the transformation parameters, because it is not intended as a production application, but rather serves as a proof of concept. To enable its cross-platform operation, Odin is implemented using Java and Eclipse's Standard Widget Toolkit (SWT).

Odin's user interface is shown in Figure 6.1. After the user specifies DTD files for the source and destination schemas, these DTDs are parsed into graphs with the

[1] *Odin* is the supreme god of Germanic and Norse mythology. Among other things, he is a god of transformation and is able to change his form (and that of others) in any way he likes.

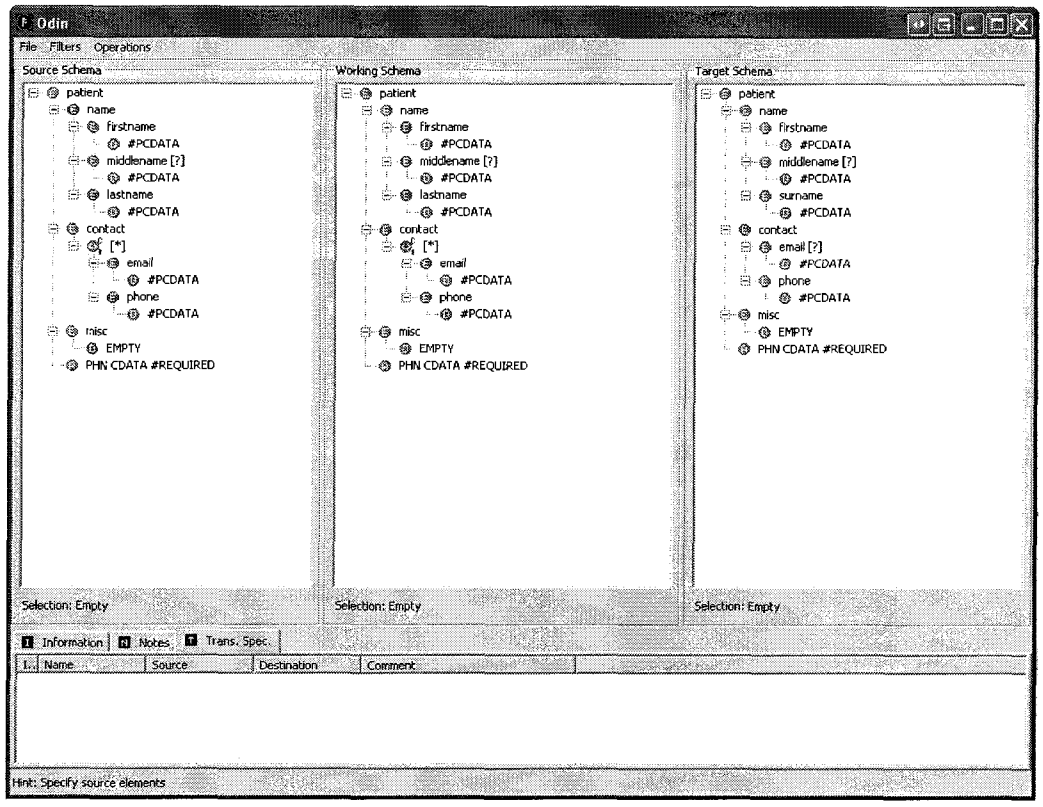


Figure 6.1. *Screenshot of Odin's user interface*

structure outlined in Section 5.1. The graphs are presented to the user as SWT Trees, where each tree node represents a node on the schema graph and is annotated with the node type and other attributes.

The user interface contains three main windows showing *source schema, working schema* and *target schema* (any of these windows can be resized/hidden by the user). Source and target windows display the source and target DTDs correspondingly and are used mainly to explore these schemas and find matches between them. The user selects and sets the parameters for the transformation on the *working schema* and then applies the required transformation to it. The *schema transformation* part [cf.

Section 5.4] of the applied transformation is used to rewrite the working schema and bring it closer to the target schema (at the beginning of the translation process the working schema matches the source schema) and the transformation is added to the recorded translation specification. This process is repeated until the working schema matches the target schema. After this match is achieved, the translation specification is produced based on the recorded sequence of transformations. This specification is later used by the *transducer* component to translate the XML documents between the source and target schemas.

6.1.2 Transducer

The *transducer* [cf. Section 3.3.2.4] is currently implemented in C# and uses Microsoft .Net Framework. It should be noted, however, that because it employs (as do other components of the *HealthMatrix* grid) Web Services to connect to the rest of the network, the specific implementation of the *transducer* can easily be changed without affecting its functionality.

The exposed (via Web Services) interfaces allow the grid administrator to configure the transducer: specify its connections to other components and define which translation specifications should be used for the translation between pairs of *schema types* [cf. Section 3.3.2.1]. Another set of functions allows transducer to interact with the grid by receiving and sending information tokens (each of these tokens carry an XML document).

After an information token is received by the transducer, it determines the schema of contained XML document. Based on this schema an appropriate translation program is selected and executed. The translated document is wrapped into a token and sent to the next component of the grid.

It should also be noted that, like all other components that use scripts (see Sections 3.3.2.3, 3.3.2.5), the transducer can be reconfigured on the fly and its transformation specifications can easily be updated if one (or more) of the schemas change.

6.2 Resolving Data Heterogeneity

In this section we explore how the various types of data heterogeneity [cf. Section 2.2] can be resolved using the transformations described in Chapter 5. These results are summarized in Table 6.1.

6.2.1 Naming Heterogeneity

Naming heterogeneity [cf. Section 2.2.1] occurs when different names are used to describe the same concept (synonyms), or when the same name is used to describe different concepts (homonyms). Once the user performing the translation has decided what the new name should be (e.g., avoiding conflicts), both of these subtypes can be resolved by applying the *rename element* [cf. Section 5.4.7] or *rename attribute* [cf. Section 5.4.8] transformation.

6.2.2 Value Heterogeneity

Value heterogeneity [cf. Section 2.2.2] occurs when the values of a particular element are represented differently in different data sources. Since it depends on the values of data elements, the only transformation capable of resolving it is *change value* [cf. Section 5.4.16]. *Change value* is a *meta-transformation* and the exact variant of it required to resolve the heterogeneity depends on a specific subtype of *value heterogeneity*:

Numeric-Numeric

Different units with fixed conversion: Since the conversion between the different units that are used to represent the values values (e.g., pounds vs. kilograms) is fixed, the *convert* variant of *change value* [cf. Section 5.4.16] can be used.

Different units with varying conversion: Since, unlike the previous case, the conversion here varies with time, at the moment this subtype of *value heterogeneity* can not be resolved by our transformations. If the translation is only one-time, then this case reduces to the previous one, however if the translation has to be repeated, then the conversion operation has to be updated every time. In the future it might be possible to incorporate this into our approach for example using Web Services.

Same units with different precision: The *convert* variant of *change value* can be used to downgrade the precision, however it is generally impossible to improve

the precision using any kind of operation. In most cases, however, the precision of the values doesn't affect the data interchange (although it might have to be taken into account in computations involving said data) and doesn't have to be resolved.

String-String

Value synonyms and value homonyms: In both of these cases we would need to replace the old values with the corresponding new ones, thus it can be resolved by the *mapping* variant of *change value*. The handling of this subtype of *value heterogeneity* could be further improved if we allow access to external resources (e.g., UMLS [24] and LOINC [18] to map the medical terminology), however at the moment it is beyond our capabilities.

Different formats Just as in the case of different units, this can be resolved by applying the string equivalent of *convert* variant of *change value*.

String "precision" The instances of this subtype can vary considerably. The simple cases, such as omitted prefix/suffix can be resolved by *concatenate* variant of *change value*. While it is possible to resolve the more complex cases, such as words being shortened, etc. by specifying the complete mappings between the shortened and complete word forms and then using *mapping* variant of *change value*, in general it is not feasible. The better solution in the future would be to employ some kind of dictionary lookup.

Numeric-String As we have said before, while the values might have different types in XML Schema, in the XML documents themselves everything is represented as text, thus this heterogeneity subtype doesn't need to be resolved.

6.2.3 Content Differences

In order to resolve the *content differences* [cf. Section 2.2.3], we need to correctly set the values of elements (or attributes) that are not directly represented in a data source. In the case of *missing data* it is impossible. In other cases (*implicit* or *derivable data*), the correct values of these elements could be derived by the user based on the domain knowledge and/or other elements and then set using the *Change value* [cf. Section 5.4.16] transformation.

6.2.4 Semantic Heterogeneity

Most subtypes of the *semantic heterogeneity* [cf. Section 2.2.4], such as *what the data represents*, *context in which data is captured* and *Meaning of* NULL do not affect the translation directly, but rather have to be explored and watched for by the user that creates the translation specification, in order for him/her to come up with the adequate matches between the schemas.

Data granularity however, requires the values of the elements/attributes to be mapped from one set of values to another. This can be accomplished using the *mapping* variant of *change value* [cf. Section 5.4.16] transformation.

6.2.5 Data Model Heterogeneity

Both subtypes of the *data model heterogeneity* [cf. Section 2.2.5] (*storage policy differences* and *differences in constraints*) require that only parts of the documents be translated (e.g., omit any record older than 10 years). As such, they require the conditional processing of some elements. This is not currently handled by our approach, however it seems to be a promising direction for future research. It is also worth noting that in our *HealthMatrix* project this issue can be mitigated by pre-filtering the records at the SFE [cf. Section 3.3.1] level.

6.2.6 Information Capacity Heterogeneity

Missing elements/attributes [cf. Section 2.2.6] are resolved by creating the empty instances of such elements/attributes using *create element* [cf. Section 5.4.2] and *create attribute* [cf. Section 5.4.4] transformations respectively. After these instances are created, the problem is reduced to *content differences* [cf. Section 2.2.3] heterogeneity type and is resolved as described in Section 6.2.3.

Different cardinality [cf. Section 2.2.6] can be resolved by the *change quantifier* [cf. Section 5.4.14] transformation. In the case where a group of elements has a different cardinality in different schemas, the *convert to group* [cf. Section 5.4.11] and *flatten group* [cf. Section 5.4.12] might have to be applied as well.

6.2.7 Structural Heterogeneity

Since all subtypes of *structural heterogeneity* [cf. Section 2.2.7] are usually intertwined, in general it requires the application of all of the structure-modifying transformations to resolve. Such transformations are: *create element* [cf. Section 5.4.2], *delete element* [cf. Section 5.4.1], *create attribute* [cf. Section 5.4.4], *delete attribute* [cf. Section 5.4.3], *move content node* [cf. Section 5.4.9], *move attribute* [cf. Section 5.4.10], *convert to group* [cf. Section 5.4.11], *flatten group* [cf. Section 5.4.12] and *fold over ID/IDREF* [cf. Section 5.4.15].

In addition to that the *element/attribute* [cf. Section 2.2.7] subtype is resolved using *convert element to attribute* [cf. Section 5.4.6] and *convert attribute to element* [cf. Section 5.4.5]. *Single element/multiple elements* [cf. Section 2.2.7] subtype usually requires application of *change value* [cf. Section 5.4.16] (most likely the *concatenation* variant) transformation in order to be resolved.

6.3 Palliative Care Data Mapping Case Study

In order to evaluate our approach to data translation, we have entered into a collaborative case study on the *palliative care*[2] *data mapping* with researchers from the University of Victoria School of Health Information Science.

The goal of this case study is to create research copies of five different palliative

[2] *Palliative care* refers to any form of medical care or treatment that concentrates on reducing the severity of the symptoms of a disease or slows its progress rather than providing a cure. It aims at improving quality of life, and particularly at reducing or eliminating pain.

	R	PR	NR	N/A
Naming Heterogeneity				
Naming Synonyms	√			
Naming Homonyms	√			
Value Heterogeneity				
Fixed conversion	√			
Varying conversion		√		
Different precision	√			
Value synonyms	√			
Value homonyms	√			
Different formats	√			
String "precision"		√		
Numeric-String				√
Content Differences				
Implicit data		√		
Derivable data		√		
Missing data	√			
Semantic Heterogeneity				
What the data represents				√
Context of capture				√
Data granularity	√			
Meaning of *NULL*				√
Data Model Heterogeneity				
Storage policy differences			√	
Differences in constraints			√	
Information Capacity Heterogeneity				
Missing elements	√			
Different cardinality	√			
Structural Heterogeneity				
Element/Attribute	√			
Single/multiple element(s)	√			
Aggregation conflict	√			
Generalization conflict	√			

R: Resolvable
PR: Partially Resolvable
NR: Not Resolvable
N/A: Doesn't apply to the translation specification process.

Table 6.1. *Resolving Data Heterogeneity: Summary*

care databases from Calgary, Halifax, Toronto, Queens and Victoria Hospice Society (VHS) and enable data interchange between them in order to be able to analyze the global palliative data (as well as variations between local implementations) and also eventually map such data to the Palliative Data Set (PDS). PDS is a set of 25 data elements that was developed by Craig Kuziemsky from the School of Health Information Science in 2002. The objective of the PDS is to provide a common data set to which palliative care data elements from various sites could be mapped.

The *HealthMatrix* project developed by our research group is well suited to provide the data interchange between these databases. The configuration of the *HealthMatrix* network for this case study is described in detail in [50]. In this thesis we concentrate on the issues concerning data translation between the palliative data sources.

The data has to be translated between each of these four data sources:

Calgary: 10 tables, 117 fields

Halifax: 49 tables, 208 fields

Toronto: 57 tables, 251 fields

Queens: 15 tables, 146 fields

and the VHS palliative database (3 separate databases, 68 tables, 401 fields). As visible from their structure, these databases are quite dissimilar, however they contain similar data, albeit augmented with information specific to how the palliative program is run at each of these hospitals. All these databases were originally implemented in Microsoft Access, but after wrapping them using the *HealthMatrix*'s SFE [cf. Sec-

tion 3.3.1], their data became available as XML documents. Unfortunately, at the moment *HealthMatrix* can not automatically produce the DTDs or XML Schemas for the wrapped data and completely handcoding them is infeasible, thus the completion of this case study has been postponed until such schemas can be produced.

However, in order to test our translation approach on the real-world data from this case study, we have handcoded parts of the schemas and applied it to them. One of the examples is shown in Figure 6.2. It displays the XML equivalent of the **PT** table from the Halifax palliative care database and a corresponding XML structure of the VHS database. The links between matching elements are shown as solid lines and the primary-foreign key relations are shown as dashed arrows. The sequence of transformations required to translate the corresponding XML documents can be outlined as follows:

1. Copy *Patient/VHS_ID* and convert it into an *IDREF* attribute of *Patient.*
2. Convert *Program/VHS_ID* into an *ID* attribute of *Program.*
3. Convert *Program/FAM_PHYS_ID* into an *IDREF* attribute of *Program.*
4. Convert *Doctors/FAM_PHYS_ID* into an *ID* attribute of *Doctors.*
5. Fold *Program* into *Patient* over *VHS_ID* attribute.
6. Fold *Doctors* into *Patient* over *FAM_PHYS_ID* attribute.
7. Delete all elements under *Program* and *Doctors* (they are not needed anymore, because their copies are in the *Patient* already.

8. Rename root element *Patient* into *PT*.

9. Create *PT/FamDocName* element.

10. Set the value of *PT/FamDocName* to be the concatenation of the *PT/DSURNAME* and *PT/DFIRSTNAME*.

11. Delete *PT/DSURNAME*, *PT/DFIRSTNAME* and *VHS_ID* attribute.

12. Rename all of the original elements from VHS to correspond to their matches in the Halifax database (as shown in Figure 6.2).

13. Create all elements that did not have a VHS counterpart.

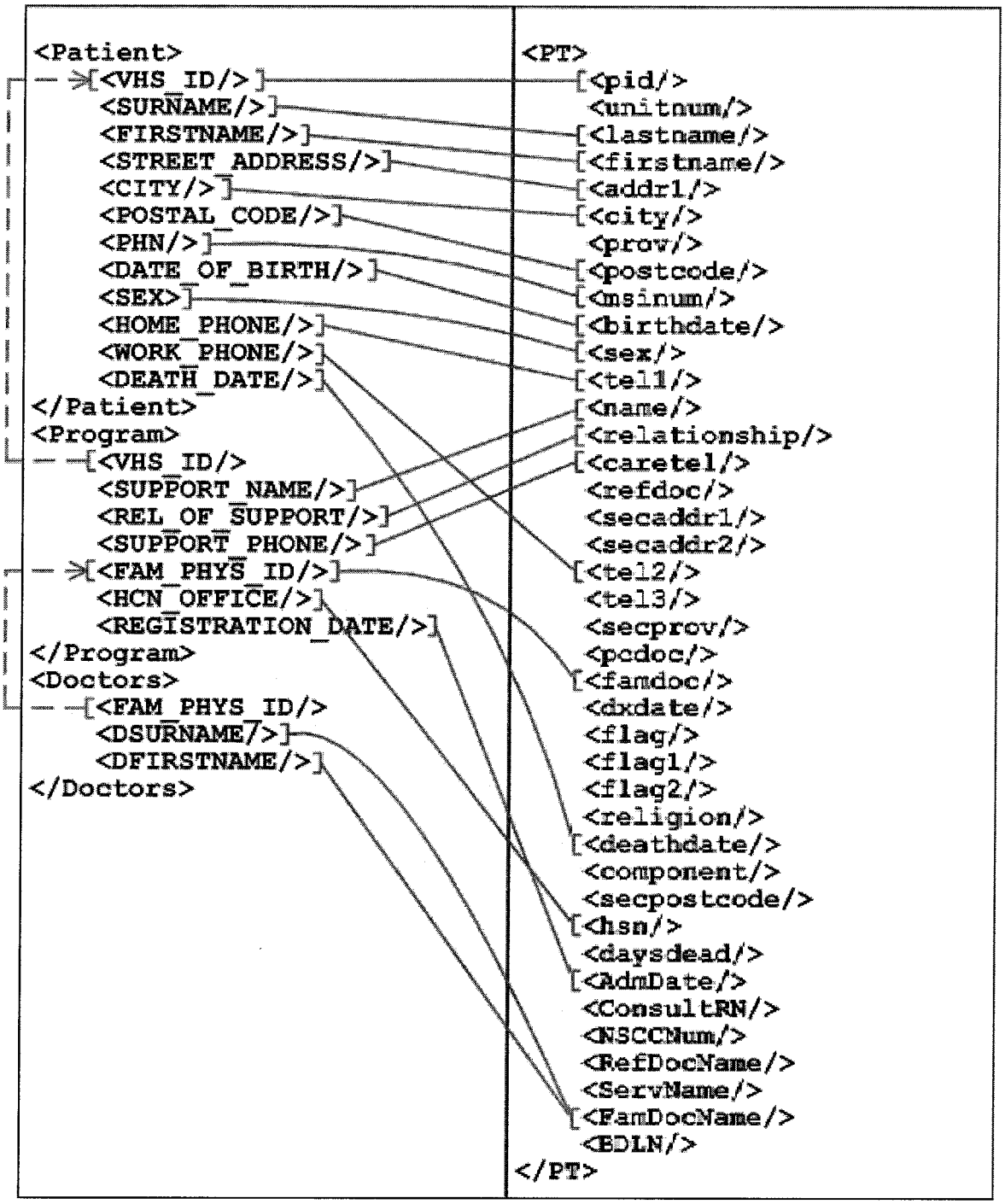


Figure 6.2. *Matches between elements from the PT table of the Halifax palliative care database and the corresponding parts of the VHS database.*

Chapter 7

Conclusions

7.1 Summary

The amount of electronic medical data (ranging from the information about individual patients to the practice guidelines and even human genome data) that had appeared in the last decade is enormous. However, there exists a big gap between the potential and the realized value of such data, mainly because of its spread over a large number of isolated medical information systems. Thus, enabling data interchange between medical information systems would improve the quality of healthcare services and even allow medical organizations to offer new services that were impossible or impractical before. Unfortunately, these information systems are highly heterogeneous and, in order to exchange data between them, the data first has to be translated into an acceptable format.

The main objective of this thesis was to develop an approach to specifying the translation of heterogeneous data. This transformation specification can be used (for example by the *transducer* component of our *HealthMatrix* project) to translate the documents between the aforementioned heterogenous data sources.

The main principle behind our approach (unlike most of the existing approaches that are based on the set of mappings between the data schemas) is to define the translation specification as a sequence of transformations that have been formally defined in both their specification as well as the effect they have on the information capacity of the schemas.

7.2 Contributions

To summarize, the main contributions of this thesis are:

Data Heterogeneity Classification: We have developed (and presented in Section 2.2) a detailed classification of types of data heterogeneity. This classification tries to abstract from the database technologies (and/or their specific implementations) and instead focuses on issues such as element names and values, semantics of the data, and its structure.

***HealthMatrix* Project:** The goal of the *HealthMatrix* project (developed by our NetLab research group) was to act as a middleware, that can facilitate data and service integration between medical information systems by connecting them into (potentially) large scale distributed networks. In Chapter 3 we have listed the requirements for such system as well as outlined its architecture and described individual components that are used to build the aforementioned health information networks.

Transformation-based Approach: The main contribution of this thesis is our

transformation-based approach to resolving data heterogeneity. Unlike the majority of existing approaches that specify the transformation between two heterogeneous data representations as a set of mappings between the elements of these representations, we proposed to do so as a sequence of formally defined transformations with well-known properties. In Chapter 5 we outlined the advantages and disadvantages of the existing approaches, and proposed our solution. Chapter 4 shows the set of transformations that we proposed as well as their properties. We also tested our approach [cf. Chapter 6] from both a theoretical (by evaluating it with respect to the data heterogeneity classification) and a practical perspective (by applying it to the *palliative care data mapping* case study).

Other Contributions: More minor contributions of this work include:

- Designing a graph representation for the XML DTDs.
- Bridging our approach with a more traditional mapping approach, by showing how a mapping can be constructed from a sequence of transformations.
- Using *history graph* idea from [44] to calculate the dependencies between elements and to show which elements (and transformations) influence others.
- Proposing several methods of optimizing the recorded sequence of transformations in order to improve the translation performance.

7.3 Future Work

The following are possible research directions that could be useful for improving our approach as well as current prototypes:

XML Schema: Current graph representation for the document schemas (and the prototype implementation as well) is based on XML DTDs. With the XML Schema growing in popularity and being used to define schemas for more and more documents, it would be useful to expand our approach in that direction.

Semi-automatic Schema Matching: As mentioned in Chapter 5, current research in the creation of XML document translators is divided into two directions: schema matching (i.e., how to determine the corresponding elements between two schemas) and translation specification (i.e., how to simplify the coding of the XML document translators). So far our research concentrated on the second problem and, while we contend that the completely automatic schema matching is too unreliable and error-prone to be applied to such domains as healthcare, the semi-automatic approach (where the matches are only *suggested* to the user), can be useful, especially when the schemas are complex.

While we did not spend considerable time researching this issue, it seems that the nature of our approach is conducive to semi-automatic schema matching, since at every step of the process it is effectively reduced to finding matches between a target schema and a modified source schema (that becomes closer to

the target schema at every iteration).

Optimization: In Section 5.5.3, we proposed several simple optimizations that can be applied to the recorded sequence of transformations. While the described optimizations are useful, they are quite simple and rely on the existence of complementary or inverse transformation in the sequence as well as on the fairly strict conditions imposed on their inputs/outputs. With further research it might be possible to find other useful (and possibly more complex) optimizations.

***Change Value* Transformation:** *Change value* is a *meta-transformation*, since it relies on an internal operation in order to compute a new value for the element. In this thesis, we described several such operations. Defining more of these operations (and possibly enabling the user to write his or her own) might be useful in resolving more heterogeneity types or simplifying the resolution of already resolvable ones.

External Lookup: Currently all information about the transformation is contained within the transformation specification. While this is acceptable in most cases, it can lead to overly complex transformation specifications in cases where a lot of mapping from one set of values to the other is involved. In these cases it is desirable to be able to look up online resources such as medical terminology repositories or dictionaries. One of the possible ways to achieve this is by using Web Services, however the impact of doing so has to be researched further.

Conditional Processing: As mentioned in Section 6.2.5, in order to resolve some of the heterogeneity types we need to apply our transformations conditionally (e.g., based on the values of elements/attibutes). At the moment it is not supported, however, it seems to be a desirable research direction.

Instance-level Information Capacity: As explained earlier (cf. Section 5.3.1), the properties of the transformations can be used to determine the effect of the translation on the information capacity of the schema. However, in some cases, this effect can be determined with greater precision by looking at the translated document itself. For example, if an element is defined in the original schema to occur zero-or-more times and in the target schema one-or-more times, such a transformation would be *information capacity reducing* in general. However, if the element actually occurred at least one time in the source document, then this instance of the transformation would not affect the information contained in the document. In the future, the transducer component can be enhanced to determine this fine-grained effect of the transformations on the actual data.

XSLT vs. Non-XSLT: Currently, we use XSLT programs to define the instance transformation part of our transformations. So far, it seems that XSLT is sufficiently powerful for this task. However, it is possible that our approach can be improved by employing other means of transforming the XML documents. This issue has to be researched and evaluated further.

Bibliography

[1] "Canada Health Infoway," http://www.canadahealthinfoway.ca/.

[2] "Document Object Model (DOM)," http://www.w3.org/DOM/.

[3] "The Extensible Stylesheet Language Family (XSL)," http://www.w3.org/Style/XSL/.

[4] "HL7 Reference Information Model (RIM)," http://www.hl7.org/library/data-model/RIM/modelpage_mem.htm.

[5] "Hyper Text Markup Language (HTML)," http://www.w3.org/MarkUp.

[6] "University of Washington XML Data Repository," http://www.cs.washington.edu/research/xmldatasets/.

[7] "VisualXSLT," http://www.activestate.com/Products/Visual_XSLT/.

[8] "XML.org DTD and XML Schema Repository," http://www.xml.org/xml/registry.jsp.

[9] "GML Starter Set," http://publibz.boulder.ibm.com/cgi-bin/bookmgr_OS390/BOOKS/DSM05M00/, 1991.

[10] "Extensible Markup Language (initial draft)," http://www.w3.org/TR/WD-xml-961114.html, 1996.

[11] "Extensible Stylesheet Language Transformations (XSLT)," http://www.w3.org/TR/xslt, 1999.

[12] "XML Schema," http://www.w3.org/XML/Schema, 2001.

[13] "XSLerator," http://www.alphaworks.ibm.com/tech/xslerator, 2001.

[14] "XSLWiz," http://www.induslogic.com/products/xslwiz.html, 2001.

[15] "Unified Modeling Language (UML) specification v. 1.5," http://www.omg.org/technology/documents/formal/uml.htm, 2003.

[16] "BizTalk Mapper," http://www.microsoft.com/biztalk/, 2004.

[17] "Extensible Markup Language (XML)," http://www.w3.org/TR/REC-xml/, 2004.

[18] "Logical Observation Identifiers Names and Codes (LOINC®)," http://www.loinc.org/, 2004.

[19] "MapForce 2004," http://www.xmlspy.com/products_mapforce.html, 2004.

[20] "MarrowSoft <Xselerator>," http://www.marrowsoft.com, 2004.

[21] "Omnimark 7.1.2," http://developers.omnimark.com/documentation/index.htm, 2004.

[22] "StylusStudio XML Mapping Tools," http://www.stylusstudio.com/xml_mapper.html, 2004.

[23] "TagFree X2X," www.tagfree.com/english/product/, 2004.

[24] "Unified Medical Language System," http://www.nlm.nih.gov/research/umls/umlsmain.html, 2004.

[25] "Whitehill <XSL>Composer," http://www.whitehill.com, 2004.

[26] "XML Spy," http://www.xmlspy.com/products_ide.html, 2004.

[27] Sihem Amer-Yahia, Mary Fernandez, Rick Greer, and Divesh Srivastava, "Logical and physical support for heterogeneous data," in *Proceedings of the eleventh international conference on Information and knowledge management*, pp. 270–281, ACM Press, 2002, ISBN 1-58113-492-4, doi: http://doi.acm.org/10.1145/584792.584839.

[28] Patric Arnold, Iryna Bilykh, Yury Bychkov, David Dahlem, Jens Jahnke, Barbara Kursawe, Craig Kuziemsky, Francis Lau, and Adeniyi Onabajo, "Society Information Grids," in *Proceedings of the 2003 NET.ObjectDays Conference*, September 2003.

[29] Alexandru Berlea and Helmut Seidl, "fxt – A Transformation Language for XML Documents," *Journal of Computing and Information Technology (CIT), Special Issue on Domain-Specific Languages*, 2001.

[30] A. Boukottaya, C. Vanoirbeek, F. Paganelli, and O. Abou Khaled, "Automating XML documents transformations: a conceptual modelling based approach," in *Proceedings of the first Asian-Pacific conference on Conceptual modelling*, pp. 81–90, Australian Computer Society, Inc., 2004.

[31] Petr Cimprich, "Streaming Transformations for XML," in *Proceedings of the O'Reilly Open Source Convention'02*, 2002.

[32] James Clark, "Comparison of SGML and XML," http://www.w3.org/TR/NOTE-sgml-xml.htm, 1997.

[33] C. J. Date, "Three-valued logic and the real world," *InfoDB*, Winter 1989.

[34] R. H. Dolin, L. Alschuler, C. Beere, P. V. Biron, S. Lee Boyer, D. Essin, E. Kimber, T. Lincoln, and J. Mattison., "The HL7 Clinical Document Architecture," *Journal of American Medical Information Association*, 8(6), pp. 552–569, Nov/Dec 2001.

[35] H. T. El-Khatib, M. H. Williams, L. M. MacKinnon, and D. H. Marwick, "A framework and test-suite for assessing approaches to resolving heterogeneity in

distributed databases," *Information and Software Technology*, 42(7), pp. 505–515, May 2000.

[36] Ramez Elmasri and Shamkant Navathe, *Fundamentals of database systems*, Addison-Wesley, 1994.

[37] Charles Goldfarb, *The SGML Handbook*, Oxford University Press, 1991.

[38] Charles Goldfarb, Edward Moshere, and Theodore Peterson, "An Online System for Integrated Text Processing," in *Proceedings of American Association for Information Science*, volume 7, pp. 147–150, 1970.

[39] Horace Hart, *Rules for Compositors and Readers, which are to be observed in all cases where no special instructions are given*, Clarendon Press, Oxford, 1893.

[40] Advisory Council on Health Infostructure, "Canada Health Infoway: Paths to Better Health," Technical report, Office of Health and the Information Highway, Health Canada, February 1999, `http://www.hc-sc.gc.ca/ohih-bsi/pubs/1999_pathsvoies/info_e.html`.

[41] F/P/T Advisory Committee on Health Infostructure, "Blueprint and Tactical Plan for a Pan-Canadian Health Infostructure," Technical report, Office of Health and the Information Highway, Health Canada, December 2000, `http://www.hc-sc.gc.ca/ohih-bsi/pubs/2000_plan/plan_e.html`.

[42] F/P/T Advisory Committee on Health Infostructure, "Tactical Plan for a Pan-Canadian Health Infostructure," Technical report, Office of Health and the Information Highway, Health Canada, November 2001, `http://www.hc-sc.gc.ca/ohih-bsi/pubs/2001_plan/plan_e.html`.

[43] Richard Hull, "Relative information capacity of simple relational database schemata," *SIAM Journal of Computing*, 15(3), pp. 856–886, August 1986.

[44] Jens H. Jahnke, Wilhelm Schäfer, Wadsack Wadsack, and Albert Zündorf, "Supporting iterations in exploratory database reengineering processes," *Science of Computer Programming*, 45(2–3), pp. 99–136, November/December 2002.

[45] H.-J. Kreowski and G. Rozenberg, "On Structured Graph Grammars," *Information Sciences*, 52, pp. 185–210,221–246, 1990.

[46] K. Jensen L.M. Kristensen, S. Christensen, "The Practitioner's Guide to Coloured Petri Nets," *International Journal on Software Tools for Technology Transfer*, 2, pp. 98–132, 1998.

[47] R. J. Miller, Y. Ioannidis, and R. Ramakrishnan, "The Use of Information Capacity in Schema Integration and Translation," in *Proceedings of the Nineteenth International Conference on Very Large Data Bases (VLDB)*, pp. 120–133, Dublin, Ireland, August 1993.

[48] T. Murata, "Petri Nets: Properties, Ananlysis and Applications," in *Proceedings of the IEEE*, volume 77, pp. 541–580, April 1989.

[49] Young-Kwan Nam, Joseph Goguen, and Guilian Wang, "A Metadata Integration Assistant Generator for Heterogenious Distributed Databases," in *Proceedings of the International Conference on Ontologies, Databases and Applications of Semantics for Large Scale Information Systems*, LNCS, pp. 1332–1344, Springer, 2002.

[50] Adeniyi Onabajo, *Grid Federation Envelope (GFE): Federating Medical Information Systems*, Master's thesis, University of Victoria, 2003.

[51] Emmanuel Pietriga, Jean-Yves Vion-Dury, and Vincent Quint, "VXT: a visual approach to XML transformations," in *Proceedings of the 2001 ACM Symposium on Document engineering*, pp. 1–10, 2001.

[52] Paul Prescod, "Introduction to DSSSL," `http://www.prescod.net/dsssl/`, 1997.

[53] Miller R., Haas L., and Hernandez M., "Schema Mapping as Query Discovery," *The VLDB Journal*, pp. 77–88, 2000.

[54] Erahrd Rahm and Philip Bernstein, "A survey of approaches to automatic schema matching," *The VLDB Journal*, 10, pp. 334–350, 2001.

[55] Roy J. Romanov, Q.C., "Building on Values: The Future of Health Care in Canada," Technical report, Commission on the Future of Health Care in Canada, November 2002, `http://www.healthcarecommission.ca/pdf/HCC_Final_Report.pdf`.

[56] G. Rosenberg (Editor), *Handbook of graph grammars and computing by graph transformations*, World Scientific Publishing, 1997.

[57] A.P. Sheth and J. Larsen, "Federated Database Systems for Managing Distributed, Heterogeneous and Autonomous Databases," *ACM Computing Surveys: Special Issue on Heterogeneous Databases*, 22(3), pp. 183–236, 1990.

[58] S.Krishnamurthi, K.Gray, and P.Grauke, "Transformation-by-example for XML," in *Proceeding of the Second International Workshop on Practical Aspects of Declarative Languages (PADL00)*, pp. 249–262, 2000.

[59] Hong Su, Harumi A. Kuno, and Elke A. Rundensteiner, "Automating the transformation of XML documents," in *Proceeding of the Conference on Web Information and Data Management*, pp. 68–75, 2001.

[60] Walter Sujansky, "Heterogeneous Database Integration in Biomedicine," *Journal of Biomedical Informatics*, 34(4), pp. 285–298, August 2001.

[61] Xuerong Tang and Frank Tompa, "Specifying transformations for structured

documents," in *Proceedings of the 4th International Workshop on the Web and Databases*, pp. 67–72, 2001.

[62] Guilian Wang, Joseph Goguen, Young-Kwan Nam, and Kai Lin, "Critical points for Interactive Schema Matching," Technical Report 2004-0779, Dept. Computer and Engineering, UCSD, 2004.