The Hong Kong Polytechnic University
Department of Land Surveying and Geo-Informatics

An XML-based Model-driven Framework for Geodata Exchange

Zhu Xu

A thesis submitted in partial fulfillment of the requirements for the
Degree of Doctor of Philosophy

Apr. 2006

CERTIFICATE OF ORIGINALITY

I hereby declare that this thesis is my own work and that, to the best of my knowledge and belief, it reproduces no material previously published or written, nor material that has been accepted for the award of any other degree or diploma, except where due acknowledgement has been made in the text.

_____ (Signed)

Zhu Xu (Name of student)

Abstract

Geospatial data (*geodata* in short) sharing is of great importance to promote and facilitate Geographic Information System (GIS) application. The adoption of numerous geodata formats and the existence of semantic discrepancies make geodata sharing a difficult task. Geodata translation, the transformation of geographic information from one data format to another and from one conceptual model to an intended conceptual model, is a key issue in geodata sharing.

The current practice is to use black-box geodata translators to convert geodata from one format to another and, when required, one or more intermediate format(s) are used to reduce the number of translators to be developed. These translators are hard coded and cannot be customized to reconcile semantic differences between source data and the intended data representation. A number of geodata transfer standards have been developed, of which each was intended to serve as *the* intermediate format but they have not been as successful as expected. Moreover, to translate geodata via a standard format simply means using two black-box translators. No mechanism is provided in the process to address semantic discrepancy. The introduction of an intermediate format also makes data translation more subject to information loss.

Recognizing these two barriers, this study aims to develop techniques for automating the generation of geodata translators and enabling high-level customization of geodata translation so as to facilitate semantic reconciliation. The promising model-driven methodology and XML technology are combined to dredge the long jammed channel for geodata flow.

A model-driven framework for geodata exchange is proposed to empower the acquisition of the desired features. In the framework, a 4-layer conceptual model of data representation is adopted, which consists of *instance*, *schema*, *meta-schema* and *meta-meta-schema* layers in a bottom-up order. Each layer is assumed to have an associated formalism for representing data. Upper layer data specifies models for lower layer data. A proprietary data format is conceptualized as a specific 3-layer model of data representation and can thus be accommodated in the framework. In a layer, there is assumed to be means for specifying transformation from one model to another. Data translation is conceptualized as a top-down chain of model transformations starting from the meta-schema layer with upper layer transformation automatically guiding lower layer model transformation, thus *model-driven*. This procedure is entitled *framework*, because it can naturally accommodate various geodata formats without significant restrictions and is open to legacy, current and forthcoming systems. In addition to automating translator generation, it makes customization of translation possible and easy, both on a dataset-by-dataset basis and on a format-by-format basis.

XML-based implementation of the proposed framework has been explored, prototyped and tested. XML (Extensible Markup Language) is adopted as the common syntax at instance layer. W3C XML Schema is adopted as the common and ultra formalism for specifying meta-schemas and schemas. On this basis, a means for declaratively specifying translation rules between meta-schemas (and schemas) has been devised and its associated processor developed. Meta-schema translation specifications can be processed automatically to generate enhanced schema translators, which can produce associated schema mappings in addition to translating source schemas to target schemas. Reconciliation of semantic discrepancy, if needed,

can be performed by human operators through customizing schema mappings in a declarative way. Schema mappings (customized or not) can be processed automatically to generate data instance translators.

Recommendations are given for further studies on completing the developed techniques and incorporating ontology technology in the proposed framework.

Acknowledgements

I am thankful to The Hong Kong Polytechnic University for funding my Ph.D. study.

I am full of gratitude to my supervisors, Prof. Yuk Chueng Lee and Prof. Yong Qi Chen. Prof. Lee led me to this interesting subject of geodata exchange. He enlightened me on the hidden challenges that I could have otherwise missed, as many others had. It was a misfortune for me to lose his guidance and friendship. Luckily, Prof. Chen took over and has been equally helpful. Prof. Chen has been extra supportive in those frustrating times and has always been understanding, inspiring and informative.

I am greatly indebted to Prof. Zhilin Li, who has “taken care of” me at all times. Without his all-around support, I could not have accomplished my study.

I owe much to Prof. Wenxi Liu (Southwest Jiaotong University, China) for his guidance, encouragement and support.

Special thanks are due to Dr. Wu Chen for his encouragement and the pleasant chats we had together. Prof. Xiao Li Ding has always shown his support to me in various ways. I thank Dr. Lilian S. C. Pun, Mr. Geoffrey Y. K. Shea and Prof. Esmond C. M. Mok for sharing their GIS expertise with me. Other LSGI (Department of Land Surveying and Geo-Informatics, The Hong Kong Polytechnic University) staff has also shown their friendliness to me. Sincere thanks to them all.

I appreciate the invaluable friendship and help from my study or work mates in LSGI. They include Dr. Guoxiang Liu, Prof. Dawei Zheng, Dr. Congwei Hu, Dr. Zhicai Luo, Dr. Zhiwei Li, Shan Gao, Dr. Tao Chen, Michael Poon, Tak Kay Ho, Yip Kong

Cheung, Carol Ng, Ida Cheung, Alice Cheung, Dr. Tracy Cheung, Dr. Ahmed Shaker, Ching-Shing Lau, Stanley Leung, Kenneth Yau, Dr. Min Deng, Shengyue Ji, and many more.

Table of Contents

List of Figures

List of abbreviations

AI	Artificial Intelligence
API	Application Programming Interface
CAD	Compute-Aided Design
CWM	Common Warehouse Meta-model
DBMS	Database Management System
DIGEST	Digital Geographic Information Exchange Standard
DTD	Document Type Definition
EBNF	Extended Backus-Naur Form
E-R	Entity-Relationship
FME™	Feature Manipulation Engine
GIS	Geographic Information System
GML	Geographical Mark-up Language
GUI	Graphic User Interface
HTML	Hyper-Text Mark-up Language
MDA	Model-Driven Architecture
MIF	MapInfo Interchange File
MOF	Meta Object Facility
NSMA	National Surveying and Mapping Agency
ODL	Object Definition Language
OGC	Open Geospatial Consortium
OMG	Object Management Group
OODBMS	Object-Oriented DBMS
ORDBMS	Object-Relational DBMS
SAIF	Spatial Archive and Interchange Format
SGML	Standard Generalized Markup Language
SDT	Syntax-Driven Translation
STDS	Spatial Data Transfer Standard

STEP	STandard for the Exchange of Product data
TIGER	Topologically Integrated Geographic Encoding and Referencing
TT	Tree Transformation
UML	Unified Modelling Language
USGS	United States Geological Survey
W3C	World-Wide Web Consortium
WWW	World-Wide Web
WXS	W3C XML Schema
XMI	XML Metadata Interchange
XMIF	XML counterpart of MIF format
XML	Extensible Markup Language
XPath	XML Path language
XSHP	XML counterpart of SHAPE format
XSL	Extensible Stylesheet Language
XSLT	XSL Transformation
XSPXSHP	The XSHP meta-schema
XSPXMIF	The XMIF meta-schema

Chapter 1

Introduction

1.1 Motivations and Objectives of this Study

Geospatial information, information about the Earth's surface and with reference to location (ISO 15046-1.2, 1998), is fundamental to many human activities. The collection of geospatial information is labour-intensive and expensive (Moellering 1991, Burrough and Masser 1998, Goodchild and Longley 1999). Given its wide usage and high cost, it is therefore highly desirable and often actually necessary that geospatial information once produced can be shared and reused in as many applications as possible (Bamberger 1995). Indeed, geospatial information is considered a public or quasi-public good in some nations (Rhind 1999).

Despite the necessity, the general sharing of geospatial information is difficult in practice. There are many factors contributing to the difficulty, including political, institutional, economical and technical ones (Alfelor 1995, Bamberger 1995, Craig 1995, Masser 1999). Policies on data openness are still among the most acute problems in some nations and in international geodata sharing (Cooke 1995). Data pricing is another major concern (Rhind 1999). In countries such as US, where problems of accessibility and cost are less serious, institutional issues become more conspicuous, especially the conflicts between data producers and data users (Craig 1995). The problem seems to be that data producers are by tradition government agencies that do not listen well to data users and do not cooperate very well (Evans

and Ferreira 1995). In addition to non-technical issues, there are technical barriers to geodata sharing.

Technically, data sharing can take place at different levels of sophistication (Berild and Wenzel 1995). Two typical approaches to data sharing are data sharing through a database system and data sharing through file exchange. The two approaches are basically complementary rather than competitive. Traditionally, geodata interchange in the form of file exchange was the only way of sharing geodata. In this data exchange approach, geodata are extracted from a source information system, exported into exchange files, transferred to data users, translated into a form usable in a target information system, and finally loaded into the target system. The main technical difficulty in data exchange is translation of data. The data translation process transforms data from a form imposed by the source system into a form that is consumable and makes sense to the target system. The emerging spatial database system software tools (Scholl and Voisard 1991, Miline et al. 1993, Oracle 1999, 2001) enable geodata sharing by means of spatial database systems. In the database approach, data sharing is achieved by building a centralized database system for shared data use within an organization (Ullman and Widom 1997). For autonomous organizations that need to cooperate closely, some form of federating database system is needed (Gupta1989, Hurson 1994). Database federation is more difficult to achieve than data (file) exchange mainly due to the autonomy of database systems, as well as various forms of heterogeneity (Bukhres and Elmagarmid 1996, Elmagarmid et al. 1999). The data exchange approach is more flexible and general as it leaves the issues involved in tightly coupled autonomous systems and tackles data issues only. This study is concerned with the technical aspects of geospatial information sharing.

In particular, the focus is on the geodata translation problem in the data exchange approach to geodata sharing.

From the technical point of view, the difficulties of geodata exchange are mainly from two sources. The first source is the lack of a de facto geodata transfer standard and the adoption of various data formats. Digital representation of geographical information, i.e. geospatial data or *geodata* in short, is the form required by computer-based information processing and has become predominant with the widespread application of Geographic Information System (GIS) technology over recent decades. Generally, the data to be shared have been produced using software and encoded in formats different from those used by data users. When data created by one software tool is to be used by other software, the data format needs to be converted. That is known as the data format problem. The general solution to the data format problem is to develop data conversion programs, i.e. data translators, which convert geodata from one format to another. However, developing geodata translators is non-trivial. Geodata formats tend to be more complex than many other kinds of digital data formats (Guptill 1990, Williams 1993). Moreover, given the widespread use of geographic information, the number of geodata formats is large as there are numerous software tools involved in geospatial information processing, each of which has its own data format. Further, data formats evolve with the evolution of their associated software and with technical improvements. GIS tools often support only a few non-native data formats for a number of reasons. Geodata standards have been developed in many countries (Moellering and Hogan 1997, ANSI SDTS 1997, GDBC SAIF 1995) and for various disciplines that produce or use geodata (IHO S-57 1996, CEN TC 278:1995). There have also been international standardization efforts (ISO 15046, DGIWG 1997). One purpose of those standards

is to serve as a software vendor-independent intermediate format so as to reduce the work of developing conversion programs. Standards can be difficult to define and put into wide practice (Salgé 1999). Unfortunately, that seems to be the case for geodata standards. On the one hand many of them have not gained enough vendor support, partly due to their complexity and the resulting difficulty in implementing data translators. For instance, the marketplace has been slow to accept and support SDTS (Arctur et al. 1998). On the other hand, they are adding to the number of data formats in use (Scholl 1990, Williams 1993). The emerging and success of software tools dedicated to geodata translation, such as FME™ (Safe Software 2005a), reflects both the diversity of data formats and the lack of well-accepted geodata standards. The FME™ software just mentioned supports, to various degrees, over a hundred data formats (Safe Software 2005d), which constitute only part of the existing formats from vendors or standardization efforts.

The other source of difficulty results from the fact that data produced needs to be used for unintended purposes. GIS technology has greatly facilitated geospatial analysis and decision support with its capability of processing diverse and large volumes of geodata in an integrated and efficient way. That makes a lot of tasks that were hard or even impossible to perform in traditional ways now relatively easy or feasible (Goodchild and Longley 1999). At the same time, this leads to an unprecedented need for geodata sharing (Masser 1999). However, diverse geodata have by tradition been produced for their respective specific use without taking into account the needs of other potential applications (Smith and Rhind 1999). When data are to be used for unintended purposes, semantic heterogeneity becomes exposed. Semantic heterogeneity refers to disagreement about the meaning, interpretation or intended use of the same or related data (Sheth and Larson 1990, Heiler 1995). On

the one hand, there is much work required on the institutional and economical aspects of geodata production to make geodata more versatile (Craig 1995, Sperling 1995) as data sharing is not possible if the semantics of source data are fundamentally different from the semantics of target applications. On the other hand, from a technical point of view it is impossible to produce geodata that exactly meet the requirements of all applications because applications are unlimited and shareable geodata live longer than individual applications (Burrough and Masser 1998). Generally, data need to be transformed and made consistent with the application semantics before they can be properly used. Such transformation is loosely referred to as semantic translation (OGC 1998), which is expected to resolve semantic discrepancies or semantic heterogeneity. Currently, semantic discrepancy is resolved largely in a manual way. Limited automation is carried out through ad hoc procedural programming on a case-by-case basis. There is no well-accepted mechanism in the public domain to facilitate semantic reconciliation in geodata translation.

As can be seen, geodata translation, the process of translating geographic information from one representation into another representation, is a key technical process in geodata sharing practice. Two significant technical barriers to geodata translation are the difficulty in writing geodata translators and the lack of a high-level mechanism to do semantic reconciliation. This study aims at a better understanding of the two barriers and finding ways to overcome them. In particular, the objectives is:

to develop methodology and open technology for

automated generation of customizable geodata translators

1.2 An Overview of the Approach

The approach adopted to facilitate writing geodata translators is to provide a model-driven framework to enable automatic generation of translators. Proprietary data formats can be aligned to this common framework by developing standalone straightforward converters once for a format, whereas the idea behind the support of semantic geodata translation is the use of explicit and declarative schema mapping or transformation, which can be easily specified or customized to resolve schematic discrepancies.

This study is based on a communicational perspective of geodata translation. Two kinds of participants are distinguished in the exchange of geographic information, the computer system and the human operator. Data format conversion as a means of communication between computer systems is distinguished from semantic data translation, which addresses not only syntax and semantics of data language but also the meaning of data and thus promotes the mechanical data exchange to meaningful information exchange. In this study, language is considered as the tool of representing and conveying the meaning used by communication participants. Two general categories of languages are involved, natural language and data language, and two levels of data meaning exist, namely formal meaning and sensible meaning. Data format conversion addresses the transformation of formal meaning espoused in different data languages while semantic data translation addresses the gap between the two levels of meaning and generally necessitates human intervention.

As semantic data translation cannot be fully automated, the key to facilitate semantic data translation is then to provide generic tools to assist human operators to resolve semantic discrepancies. A practical approach to automating the writing of geodata

data translators and facilitating the reconciliation of the semantic discrepancy is proposed. In this approach, the syntax of data languages or formats are aligned to and/or specified in a common framework of geodata translation (see below for explanation) on a once-a-format basis. The semantic relationship between different data languages or formats is specified using some high-level declarative language on a format-by-format basis. With these high-level specifications, a default target schema can be derived automatically from an input schema together with the associated semantic relationship between them. The semantic relationship between different datasets is specified in the case of prescribed target schema, or customized in the case of generated target schema, by human operators using some high-level declarative language on a dataset-by-dataset basis. With these high-level specifications, geodata translators can be generated automatically.

A model-driven framework for geodata exchange has been proposed to enable automating translator generation for potentially a large number of data formats. In the framework, a 4-layer conceptual model of data modelling is adopted, which consists of *instance*, *schema*, *meta-schema* and *meta-meta-schema* layers in a bottom-up order. Each layer is supposed to have an associated formalism for representing data. Upper layer data specifies models for lower layer data. A proprietary data format is conceptualized as a specific 3-layer model of data modelling and can thus be accommodated in the framework. In a layer, there is assumed to be the means for specifying the transformation from one model to another. Data translation is conceptualized as a top-down chain of model transformations starting from the meta-schema level down to the data instance level. Furthermore, the upper layer transformation guides the lower layer transformation.

The conventional way of facilitating the implementation of geodata translators is to provide a reusable API library. However, that method necessitates laborious imperative programming. A dataset is basically a database. Database technology has the potential to greatly facilitate geodata translation by performing data translation in a declarative way and at a high level of abstraction. Unfortunately, the traditional relational database is not well suited for geodata processing. This study explores the implementation of the proposed framework with XML (Extensible Mark-up Language) technology. In XML-based implementation, XML is used as the common syntax for data instance encoding; WXS (W3C XML Schema, W3C stands for the World Wide Web Consortium) is used as the common (meta-) schema definition language; on the basis of WXS, a method for specifying (meta-) schema mapping is devised. Processors have been developed to translate (meta-) schema mappings to XSLT (XSL Transformation, XSL stands for Extensible Stylesheet Language) stylesheets.

1.3 Scope of this Study

Geodata translation can be an over complicated task if the data formats involved have quite complex spatial models. When complex data formats are involved, it generally demands sophisticated geospatial processing in addition to data organization conversion. For example, consider converting a MIF (MapInfo Corporation 1999) dataset to a SDTF Topological Vector Profile (USGS 1997) dataset. The former has no topological information while the latter topologically supports the two-dimensional manifold. It will require topological processing in the translation process. Moreover, under the title of semantic geodata translation, there is a trend to put more sophisticated spatial processing into the data translation process.

For example, the conversion of a detailed spatial representation of a feature into a simplified spatial representation of the feature generally necessitates a specially designed data structure and algorithm (Weibel 1998, Lee 2000). Some consider that as a kind of semantic geodata translation (Devogele et al. 1998, Uitermark et al. 1999). If all the potential diversity is taken into account, the outcome is likely to be over burdened if any further advance is made from the present position.

To make the problem manageable, the scope of discussion has to be defined and some assumptions about the data formats to be dealt with have to be made. This study takes data exchange as a kind of data management process rather than a kind of geospatial processing process. The study will focus on changing information representation in terms of the database. The task of data translation is then to make data ready for a program to access or to make data representation conform to or come closer to the conceptual model of a particular application. There is an argument that it is more suitable to do geospatial analysis within a GIS than at the interface. Significant changes of spatial representation in forms other than data schema change are generally not within the scope of this study. In particular, spatial data generalization, of which the second of the above two examples is an illustration, is not among the main concerns. Regarding data formats, it is assumed that the two formats involved in a particular data translation case are of the same level of topological sophistication. That assumption generally excludes building topology from this study and turns topological relationships into plain associations among spatial objects.

In addition, this study aims to develop methodology and technology, not to implement many individual geodata translators that are ready to use or to fully

implement a dedicated geodata translation system. The aim is to develop new and better methods and techniques of implementing geodata translators and of performing semantic reconciliation. Therefore, only implementation to the extent that is considered to be necessary and sufficient to show the value and feasibility of the proposed ideas will be carried out.

Finally, the discussion will be restricted to geodata that are results of feature-based spatial modelling. This is because feature-based spatial modelling is fundamentally different from field-based spatial modelling (Burrough and McDonnell, 1998). The integration of feature-based modelling and field-based modelling has long been known as an involved problem.

1.4 Thesis Organization

The rest of this thesis is organized into seven chapters. Chapter 2 investigates issues involved in geodata translation. It sets down the task of geodata translation and provides reference criteria for evaluating geodata translation techniques. In Chapter 3, a review of geodata exchange techniques is presented. It serves to relate this study to other work on data exchange and further justifies the significance of this study. Chapter 4 describes the model-driven framework of geodata exchange. Chapter 5 first introduces XML technology and then presents an overview of XML implementation of the framework. It also includes discussion on how to align proprietary data formats to XML-based implementation of the framework. Chapter 6 presents the schema mapping method devised in this study and the associated algorithms for translating schema mappings to data instance translators. Also included are case studies and comparison with related work. Chapter 7 begins with the method of specifying meta-schemas and moves on to meta-schema mapping

specification. It then addresses the problem of generating schema mapping generators from meta-schema mappings. Case studies are provided. Comparison with related work is presented. Finally, Chapter 8 summarizes this study, makes concluding remarks and gives recommendations for further studies.

Chapter 2

The Geodata Exchange Problem

This chapter examines issues involved in geodata exchange from two points of view. The first concerns data organization and the second information communication. Detailed investigations will be performed to reveal the issues to be tackled in geodata translation. These issues define the tasks to be accomplished by the geodata translation process and can serve as reference criteria to evaluate existing techniques of geodata translation as well as techniques developed in this study.

2.1 Data Organization Differences

As is known, information is represented by organized data in computers. The data organization rules define the way in which information is captured by data. Data organization means not only the physical format of data but also the underlying concepts used for data modelling. For data produced in one computer environment to be used in other environments, data organization differences have to be resolved. This is a basic problem of data translation.

2.1.1 Data Organization in Database Systems

The database approach of storing, accessing and managing data has gained dominance over the traditional data file approach in many application fields. In a database system, data is basically organized at two abstraction levels (Date 1990),

namely, the logical level and the physical level as shown in Figure 2.1. For a particular database, its logical structure is specified as a data schema using the data definition language provided by the DBMS (Database Management System). For a given DBMS, users cannot define arbitrary logical structures. To work with a database system, users must accept the logical modelling formalism (called the logical data model by many) provided by the DBMS and use its data definition language. The logical modelling formalism provides the constructs and defines rules for users to build their own logical data structures or data schemas. The relational model was once the dominant logical model formalism of DBMSs. Relational database systems provide constructs such as *table*, *field* and *record* and so on. At the same time, there are rules such as that all data must be contained within tables and that a field can only be of built-in data types and cannot be of user defined data types, etc. Most DBMSs also provide the *view* mechanism that allows users to define various views of the same logical structure of the database. Views are at the same abstraction level as the logical data structure but provide much convenience for database management.

Each DBMS has a physical data representation mechanism, often called the physical data model, which provides constructs, such as *field*, *record*, *block* and *file*, to construct a concrete physical data structure. Given a logical data structure, the DBMS takes care of its mapping to a physical data structure according to some internal *encoding rules* so that data can be put into and extracted from the database system. When users want to manipulate data, they use the provided data

manipulation language to interact with the database system according to the data schema without needing to know where the data is and how data items are physically structured and encoded. The essential function of a DBMS is to provide high-level (i.e. logical level) data accessing methods.

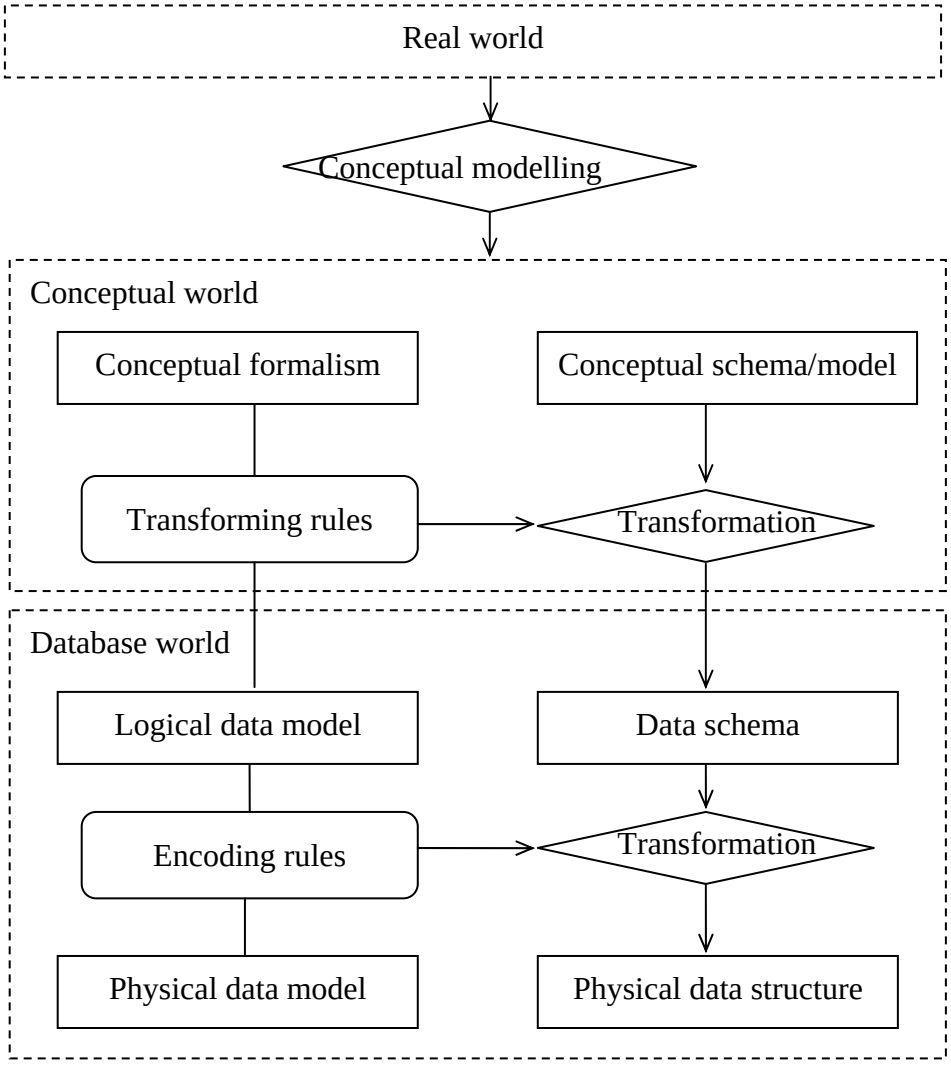


Figure 2.1 Information modelling and data representation

2.1.2 Abstraction Layers of Information Modelling

From the information modelling point of view, there is another level of abstraction in the representation of information, the conceptual level (Batini et al. 1992, Sanders 1995, Bédard 1999). A conceptual model refers to a mental model of the universe of discourse as a result of conceptual modelling or conceptualization. People think at an abstraction level higher than current computers do things. In the practice of information modelling and decision-making based on information system, it is desired that the real world is modelled at a level that is natural to human beings instead of catering to computer-oriented models. A conceptual model can only be fully described in natural languages. Due to the freeform nature of natural language description, it is necessary to formalize conceptual modelling and conceptual models. Conceptual formalisms are invented to provide the higher-level constructs and modelling rules for formal conceptual modelling. The associated conceptual schema language provides the language to formally specify conceptual models as conceptual schemas. The most popular conceptual formalisms used for conceptual modelling of information are the object-oriented model (Anstey 1998, Blaha and Premerlani 1998) and the entity-relationship (E-R) model (Batini et al. 1992). Examples of graphic conceptual schema languages are UML (Unified Modelling Language) (Fowler and Scott, 2000) and E-R diagram (Batini et al. 1992).

A given conceptual schema needs to be converted into a logical data schema that can be implemented in database systems. If the logical data formalism of the DBMS is

close to the conceptual formalism, the mapping from conceptual schema to data schema is rather straightforward. Otherwise, more sophisticated rules have to be employed for the conversion. For example, the mapping from ODL (Object Definition Language (Cattell and Barry 1997)) to C++ is straightforward while the mapping from the E-R model to a relational model needs additional care (Ullman and Widom 1997).

What needs to be emphasized here is that the abstraction level of conceptual formalisms is often higher than that of the logical data formalism provided by DBMS. During the conversion process, some semantics of conceptual schemas cannot be explicated if the logical formalism lacks some of the modelling mechanisms available in the conceptual formalism (Bekke 1992, Gray 1992). The part of the conceptual model that is lost needs to be compensated in the process of application programming. It should also be noted that there is more than one way to carry out the conversion from conceptual schemas to data schemas. Different data schemas may result from one conceptual schema due to different application purposes or representation preferences (Bekke 1992). The implementation of the state-of-the-art conceptual formalisms in database systems is a goal of database technology. The object-oriented model is the conceptual formalism that is widely accepted and now being introduced into DBMS (Bertino and Martino 1993, Lomis 1995), resulting in object-oriented DBMS (OODBMS) (Barry 1996, Loomis and Chaudhri 1998) or object-relational DBMS (ORDBMS) (Carey 1997, Oracle 1999, 2001).

2.1.3 Geodata Organization in GIS Software

In application fields, e.g. GIS and CAD, where conventional database systems cannot fulfil the data management requirements, domain-specific software tools provide some capability of data management. Usually, the software tool hides the physical details of the storing and accessing of data from the users by providing some logical data model and associated interaction or programming methods of manipulating data.

The representation of geospatial information using GIS software tools is basically the same as transaction information representation using database software tools. However, there are significant differences. First, due to the complexity of geospatial information, there is no standard logical geodata model in the spatial database field that is as theoretically solid and practically popular as the relational model in the general-purpose database field. Vendors each have their own geodata models for their GIS products. Second, there is no standard high-level data definition and manipulation language for defining and accessing a spatial database, such as the SQL language used for relational databases. There have been many efforts aimed at defining a declarative query language for spatial databases. Unfortunately, none of them has gained wide acceptance and popularity. In practice, spatial data has been accessed in two ways, the interactive way through a graphic user interface (GUI) and the programming way through product-proprietary API (Application Programming Interface). This means that the retrieval of geodata from GIS systems is basically accomplished in a procedural way. That situation is changing thanks to the efforts of

OGC (formerly Open GIS Consortium and now Open Geospatial Consortium) and the entering of major DBMS vendors to the field of spatial data management (see next chapter for a review).

2.1.4 Geodata Organization in Exchange Formats

When data go out of GIS systems to be used by other applications, it is encoded into data files according to some exchange format. The term *format* is sometimes used to refer to a general-purpose file format, such as ISO/IEC 8211 (ISO/IEC 8211:1994) and XML (Extensible Mark-up Language) (Yergeau et al. 2004). Here *data format* means a fixed combination of logical data model, encoding rules and file format (which corresponds to the term physical data model in database systems); that seems to be the meaning of *data format* referred to in most literature related to geodata exchange. From a practical point of view, a data format specifies a way of making data persistent in computer files. The encoding rule bridges the two levels of data abstraction and defines how to map logical data structures to file structures. It should be noted that a file format is not the file structure of a specific data file. Rather, it defines a model for file structures of all the files conforming to the format. Therefore, a data format will always have a way to completely recover the logical data structure from a specific file structure through hard coded rules or special records explicitly contained in the data file. Many data formats will have file header records, in which the logical data structure is stated.

It should also be noted that a) data of one logical data model can be encoded into various file formats; b) data of one logical data model can be encoded into the same file format using different encoding rules, resulting in different file structures if the file format is a general one instead of a proprietary one tightly coupled to the specific logical model; c) a general purpose file format can be used to encode data of various logical meta-data models.

As can be seen, there is no conceptual difference between data organization in database systems and that in data exchange files at the logical level. The difference lies mainly at the physical level. Physical data models of database systems (or GIS software) aim at gaining data storage and accessing efficiency (Date 1990) by being hardware dependant, using binary data encoding, using physical pointers, using index, etc. The data exchange format aims at facilitating the data exchange process by being hardware independent, using standard binary or textual encoding, using logical pointers, etc (Lee and Coleman 1990). Physical level details are not elaborated here as they are no longer a problem thanks to well-accepted physical level standards. Geodata exchange formats will now be considered in more detail.

2.1.5 Geodata Exchange Formats

Geodata exchange formats generally come from two camps. One camp is that of the software tools. There are many software tools involved in geodata processing. In addition to GIS, CAD (Compute-Aided Design) and cartographic software, which are widely used in geodata creating, editing and map making, many other software

tools have been involved in geodata processing, including those used in professions such as urban planning, facility management, transportation planning and management, environment survey and protection and so forth (Longley et al. 1999). Data exchange is needed, both among competitive products and among complementary products, because it is often necessary to use several tools to solve a complex problem and use data from more than one source. To facilitate data sharing, some of them will provide data exchange formats (ESRI 1998, Mapinfo 1999).

The other camp for geodata exchange formats results from various geodata standardization efforts. Geodata producers and users have been actively involved in geodata exchange standardization efforts. In many nations, the National Surveying and Mapping Agency (NSMA) plays an important role in geodata standardization. STDS (Spatial Data Transfer Standard) (SDTS 1997) is a United States standard and the earliest national standardization effort on geodata exchange. There have also been international geospatial information standardization efforts, globally or regionally. ISO/TC211 is the ISO technical committee that works on geospatial information. CEN/TC287 is an example from the European Committee on Standardization. DISGEST (DGIWG 1997) is the NATO (North Atlantic Treaty Organization) geodata standardization effort. The above-mentioned standards can be called general-purpose geodata standards. There have also been application-oriented standards. The S-57 (IHO S-57 1996) is a well-known international standard for exchanging hydrographical data and GDF (CEN TC 278:1995) is a European standard for exchanging road data and is undergoing the process to make it an

international standard. OpenGIS Specification is an industrial effort to achieve geoprocessing interoperability among GIS tools from diverse vendors (OGC 1998). GML (Geographical Mark-up Language) (OGC 2002) is an XML-based geodata exchange format and part of OpenGIS Specifications.

Vendors' data exchange formats often aim to be computer platform-independent and easy to implement. A general-purpose geodata exchange standard aims to be vendor-independent, comprehensive and suitable for exchanging a wide range of geodata (Berild and Wenzel 1995). Standards for an application domain often aim at providing a common conceptual model for the domain. The following observations can be made regarding the geospatial information representation mechanisms they provide.

The logical data models used are variants of an extended relational model, called a geo-relational model by some, and object-oriented model. The differences exist in the spatial data types supported and modelling constructs provided as well as the simple data types available. Geodata exchange standards usually explicitly specify their logical meta-models as the so-called application modelling rules (ISO 15046-9). Vendors usually do not clearly specify their logical models in associated documents (ESRI 1998, Mapinfo 1999). Users have to work intimately with the software to figure it out.

In vendors' data exchange formats (as well as internal data representation methods of software), it is often the case that the logical model, encoding rules and physical

model are tightly coupled. Hence, the proprietary file format cannot be used to convey data other than that conforming to their associated logical models. That is going to change with the popularity of XML. Some standard formats have used standard general-purpose file formats. For example, STDS (USGS 1997d) and DIGEST use ISO 8211 encoding while ISO 15046 (ISO 15046-18) and OpenGIS Specification use XML encoding (OGC 2002).

2.1.6 Differences in Data Organization

Based on the above discussion of data organization, it should become clear that data organization differences could exist at both the logical level and physical level. In each of these levels, there are differences in both the model and the schema/structure. That is because, given a higher-level schema, there is more than one possible representation at the lower level as discussed above. Figure 2.2 gives a summary of the categories of data organization differences.

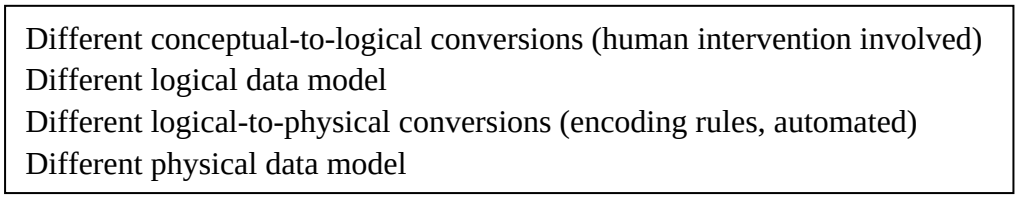


Figure 2.2 Categories of data organization differences

2.2 Semantic Heterogeneity

In addition to data organization differences among computer systems, an application may have a conceptual model of the real world that is different from that of the data

creator. Moreover, when data are extracted from an information system, their connection with the application context in which the data is embedded is cut. Receivers of external data may interpret data in a different way. That introduces what is called in the literature "semantic heterogeneity" (Sheth and Larson 1990, Heiler 1995). To achieve meaningful data exchanges, semantic differences between source and target information systems need to be addressed. This section attempts to trace the sources of semantic heterogeneity and provide an understanding as the basis for seeking a solution in later work in this study.

2.2.1 A Working Definition of *Meaning*

Semantic heterogeneity occurs when there is a disagreement about the meaning, interpretation or intended use of data (Sheth and Larson 1990). Therefore, it seems necessary to inspect what meaning is, how meaning is conveyed, the process of communication and how disagreements upon meaning may happen.

According to the American Heritage Dictionary, *meaning* is "something that is conveyed or signified", "something that one wishes to convey, especially by language" or "an interpreted goal, intent, or end". The above statements are largely from the viewpoint of communication and seem to distinguish three kinds of meaning: the intended meaning (of a speaker), the signified meaning (of a representation) and the interpreted meaning (of an audience). But it does not concern the nature of what *something* is.

It is extremely difficult to give a general definition of that *something* that is being communicated. In the case of data sharing, *meaning* of data (that *something*) is considered in this study as the sum of a conceptualization of the universe of discourse and the facts observed according to this conceptualization. We call this meaning the *sensible meaning*. The emphasis is to make this meaning independent of the language used to signify and convey it. The counterpart of sensible meaning is *formal meaning*, which is defined as the meaning of data in a certain language that can be understood and processed by the speakers of that language. When the language is a natural language and the speaker is a human being, formal meaning is the same as sensible meaning. When the language is some artificial language and the speaker is computer software, formal meaning is the part of sensible meaning that is signified or formalized and can be processed by the software. One benefit of making the distinction is that it is possible to tell the difference between a database system and an information system. The meaning level of a database system is that of the database language while the meaning level of an information system is promoted to sensible meaning due to the participation of human operators.

2.2.2 A Communication Model

From language studies (Jeffries 1998), it is known that to communicate effectively by a natural language it is necessary to *a)* grasp the grammar and a (symbolic) vocabulary of the language, *b)* be equipped with the necessary commonsense or domain knowledge to make sense of the vocabulary and *c)* -have a correct perception

of the context. To illustrate these notions, some simple examples are helpful. When reading the sentence *"Tom likes coffee"*, we are likely to interpret it as "Someone, named Tom, likes drinking coffee", if we know English to some extent. However, the statement *"An algebra is a set together with operations defined in the set that obey specified laws"* doesn't make much sense to those who are totally new to algebra. It is not the new words if any that are hard to understand but rather that one lacks the knowledge to make sense of it. Yet, the sentence *"Tom is chasing Jerry"* cannot be unambiguously interpreted without referencing a certain context. Finally, while the statement *"This spatial database is of scale 1:5000"* might be used to mean "the spatial database is created by digitizing 1:5000 paper maps", it is not a rigorous statement since the concept of scale of spatial databases has no well-accepted definition. This shows that domain knowledge needs to be built and shared to enable the precise exchange of meaning.

To identify the sources and forms of semantic heterogeneity, further discussion is based on a communication model as shown in Figure 2.3.

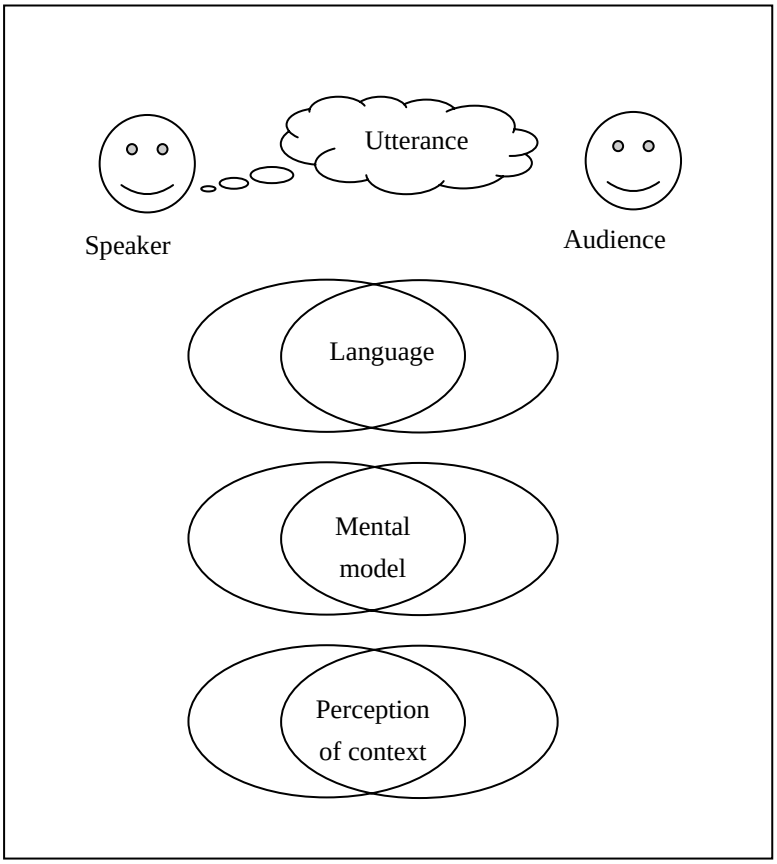


Figure 2.3 A communication model

In this model, the communication environment consists of a pair (speaker and audience) and a conversation context. Both the speaker and the audience have their own mental models about the topic and may speak in different languages. They may also have different perceptions of the context. A speaker has something (the intentional meaning) to say and develops and explains it by first representing it using his language and then issuing a piece of utterance. The audience receives the utterance and tries to understand its meaning if they know the language. First, the utterance is parsed according to the language grammar to obtain a structure of the utterance. Then, the audience applies its vocabulary and perception of the context to interpret the utterance. During the process of interpretation, the audience may need to

perform some disambiguation on the words involved, the structure of the utterance and the context. Finally, the audience applies its conceptualization to make sense of the interpretation to comprehend the meaning. The disambiguation process may be repeated many times until the audience believes the "right" meaning is obtained. It should be noted that the audience should understand the meaning of the utterance but does not necessarily agree with it.

2.2.3 Communication within and among Information Systems

Information processing employing a database system is comparable to communication using a natural language. When a query is issued against a database system, an answer is given, which is assumed to be semantically correct and complete. Let us examine how meaning is exchanged between the user and the computer system and how semantic heterogeneity is avoided in the human-computer interaction, in particular human-database interaction. In this interaction, the common language used is the database language, which is free of grammatical ambiguity. Users need to be able to use the database language. Documents have to be provided to users for them to operate on the *views* or *tables* in a meaningful way. All terms are defined including *table* names and *field* names, i.e. the vocabulary is fixed. The database system can understand the programs and processes data according to its own language and its own "mental model", i.e. the data schema. To a computer, data have only formal meanings in the sense of the database language and no intentional meanings in the sense of the specific application. In summary, in this interaction the

user adapts oneself to the database system in the communication. The user must stick to the vocabulary and data schema, because any deviation will result in error responses unless the program is informed and/or updated. In fact, the computer does not care if the audience/speaker is a human user or a remote computer as in the case of homogeneous distributed database systems. It should be stressed that a human operator is in charge of promoting the formal meaning of data to the sensible meaning that makes sense to the application or decision-making.

	human-database interaction	Database-database communication
Speaker/Audience	Human operator	Database system
Audience/Speaker	Database system	Database system
Common Language	Database language	A common database language
Common Context	That of the database system	Compromised context
Common mental model	Data schema	Integrated schema
Common vocabulary	Name of tables, fields, views etc.	Reconciled vocabulary
Common Meaning level	Formal meaning of database language	Formal meaning of the common database language

Figure 2.4 Communication setting of human-database interaction and a possible communication setting (in federated database system) for successful database-database communication

Communication would fail if a common setting of the language, the context, and the mental model cannot be reached. Misunderstanding may occur if there is any disagreement upon the communication setting that is not identified. In general, it is easy for human beings to reach a common communication setting and intentionally change to another setting, given our intelligent capability. In contrast, a common communication setting has to be set up by human operators to allow computer

systems to communicate by themselves. The communication setting of human-database communication and a possible communication setting for successful database interoperation is shown in Figure 2.4. To achieve interoperability among autonomous database systems, it is wise to use a common database language. Diverse implicit application contexts need to be made explicitly and reconciled to provide a shared context of communication. Diverse conceptual models and data schemas need also to be reconciled to reach a common model of the wider universe of discourse. That also provides the basis for employing a common vocabulary. With all those efforts of setting up a communication environment, the autonomous database systems can then interoperate automatically at the formal meaning level.

2.2.4 Sources of Semantic Heterogeneity

Based on the above discussions, sources of semantic heterogeneity in geodata exchange can be identified as shown in Figure 2.5, which includes formalization difference, conceptualization difference and differences in assumed/perceived context.

Semantic heterogeneity

|→ *Formalization heterogeneity*

| |→ *Language heterogeneity*

| |→ *Representation difference (schematic/structural heterogeneity)*

|→ *Conceptualization heterogeneity*

|→ *Context heterogeneity*

Figure 2.5 A classification of semantic heterogeneity in terms of sources

Formalization heterogeneity means the different ways of formalizing a conceptualization using data languages. It results from two factors. The first is that different data languages can be used to formally represent a conceptualization and these different languages may be of different semantic richness. For geographic information, these are geodata formats as discussed in the previous section. The second factor is that there are alternative ways to formalize a conceptualization using a language. This introduces what is called structural or schematic heterogeneity (Kim and Seo 1991).

Conceptualization heterogeneity is distinguished from formalization heterogeneity in the sense that it results from fundamental differences in modelling the real world (Lee et al. 1996). It usually means that the data is not readily suitable for a certain application (Colomb 1997).

The third source of semantic heterogeneity is related to the context of geodata collection and its use. In linguistic studies, context refers to the words around a word, phrase, statement, etc., often used to help explain or fix the meaning. The more

general meaning of context refers to the general conditions (circumstances) in which an event, action, etc. occurs (Akman and Surav 1996). For databases, Goh et al. (1994, 1999) used *context* to refer to the implicit assumptions made when an interoperating agent routinely represents or interprets data. When different agents try to cooperate, there often exists context heterogeneity that needs to be resolved. For geodata, measurement unit, reference system and projection used when creating geodata are among the most significant contextual factors.

It should be emphasized that, although conceptual heterogeneity and contextual heterogeneity are different from schematic heterogeneity, they need to be taken as schematic heterogeneity in data translation processes because the conceptual model and contextual information are at a higher abstraction level than data schema and there is no other way of implementing and processing them in the current data management technology. This leads to the argument that although conceptual and contextual heterogeneity cannot be identified by looking at data schemas, they have to be reconciled by means of transforming data structures and data values according to the change of view on the real world.

2.3 Summary

This chapter has investigated issues involved in geodata translation from two perspectives. One is data organization and the other information communication. Detailed investigations have been performed to reveal the problems to be tackled in geodata translation. From the data organization point of view, the main issues

involved are logical level differences, including different modelling constructs and semantic richness of data models, as well as different spatial data types and simple types. From the information modelling and communication points of view, there are semantic differences. These include differences in conceptualization of the real world in different applications, differences in implicit application contexts, different requirements when converting a certain conceptual model to schemas of different logical models, and different ways of converting a specific conceptual model to data schemas of a specific logical model.

As can be seen, geodata translation or data translation in general is a complicated problem. The task of geodata translators is to resolve such data organizational differences and semantic differences. In the next chapter, techniques for (geo-) data translation in particular and (geo-) data sharing in general will be reviewed with reference to these issues.

Chapter 3

A Review of (Geo-) Data Sharing Techniques

3.1 Standardization Efforts and their Limitations

GIS practitioners have long recognized the needs and difficulties of geodata sharing. In the 1980s, the main barrier was attributed to the adoption of diverse data formats. Realizing that using a standard intermediate data format could greatly reduce the number of data translators needed, many spatial data exchange standardization efforts were initiated in many countries and by many international standard organizations (Moellering 1994). For summaries of geodata exchange standardization efforts, please refer to Moellering (1991), Williams (1993), Moellering and Hogan (1997), and Salgé (1999). The theory guiding these standardization efforts is that of geodata organization.

An important achievement of the theoretical study on geodata exchange was the abstraction levels of geodata organization (Moellering 1991, Arctur 1998). Most standards adopted a multi-level abstraction of geodata, including the conceptual level, the logical level and the physical level, etc., which in general was in alignment with that of data modelling levels in the database field.

A second important idea in geodata exchange standards is the use of metadata to help interpret data and make proper use of data (Guptill 1990, Arctur et al. 1998).

Another important concept introduced from geodata standardization efforts was the separation of data and data presentation. As can be seen, many more recent standards

(in the 1990s) address spatial data exchange instead of cartographic data exchange, which the earlier (in the 1980s) standards did.

However, from today's point of view, the widespread standardization efforts of the late 1980s and early 1990s suffer from some common flaws. As observed by Mollering (1993) and Arctur et al. (1998), further fundamental research on geodata exchange is very much needed.

First, most standards adopted complex data models, which even exceeded the processing capability of most GIS software tools at that time (Scholl 1991, Williams 1993, Arctur 1998). As analysed by Moellering (1991), to reduce information loss in the process of data exchange, a standard format must accommodate the complexity of potential data formats that will use the standard format as an intermediate. Such a complexity brings difficulty in data interchange between similar GIS systems (Stigberg 1994) and also greatly raises the cost of using the standard (Arctur et al. 1998). The implementation difficulty is left to GIS vendors or users who do not need or want to deal with the full complexity of the standard. As a result, some simple data formats become much more supported and popular in practice than the official standards (National Research Council 2001).

A second limitation of geodata exchange standardization efforts is that not enough effort has been made to develop corresponding data translation technology. Being complicated but without developing associated technology, geodata transfer standards can easily become documentations. As an amendment, some standardization bodies provide basic programming support to their standards, such as the sdts++ program library of SDTS (Altheide 1996) and the SAIF development kit (GDBC 1994). However, these are low level and language-dependent API

(Application Programming Interface) for building decoders/encoders and not intended for end users, which in itself makes them less than sufficient.

Third, most standards emphasize (reasonably) the specificity of geodata exchange while to a large extent ignore the possibility and benefits of taking geodata exchange as a specialization of general electronic data exchange. This isolation of geographic information processing on the one hand impedes the integration of spatial data and non-spatial data and on the other hand impedes the adoption of general-purpose software tools for geodata exchange. Later standardization efforts, including ISO-15046 and the OGIS effort of Open GIS Consortium (OGC), recognize the necessity and benefits of the fusion of GIS and other information technologies.

Last but not least, standardization efforts assume a perfect data exchange environment, in which data producers and data users share an identical conceptual model. In practice, the assumption is normally not satisfied as every application may take a different view of the real world. This situation necessitates on the one hand that data creators produce multi-purpose geodata that suits a wide application range and on the other hand that there are mechanisms that help deal with conceptual differences in the geodata exchange process.

It is argued in this study that, given the complexity of the geodata sharing problem, both standard data formats and sophisticated data translation techniques are needed to achieve successful geodata exchange in a wide range of circumstances. Besides emphasizing the role of standard formats in geodata exchange, this study emphasizes the need and importance of sophisticated geodata translation techniques and "standard" translation methods. In addition, the widest possible use of general IT techniques is advocated.

3.2 Conventional Geodata Translators and their Associated Problems

Generally, geodata translation is a process of changing geodata from one representation to another representation that is suitable for use in the target application system. A computer program doing the job is generally referred to as a geodata translator. In current geodata translation practice, two types of geodata translators can be identified. These may be called the mechanical translator and the semantic translator, respectively. A mechanical translator converts data from one data format to another data format automatically but in a mechanical way according to some hard coded conversion rules. Most of the current geodata translators take this approach (Altheide 1992a, Altheide 1992b, Lazar 1992, Stigberg 1992, Williams 1992). In contrast, semantic translators allow custom translations to be specified (Safe Software 2005b).

3.2.1 Structure of Mechanical Geodata Translators

Figure 3.1 illustrates the components and data translation procedures of a mechanical translator between a pair of data formats. For a given data set consisting of files in the source format, a decoder reads the bytes of the file and interprets these bytes into data constructs of the source format. The constructs of the source format will be mapped to the constructs of the target format by a content translator according to some predefined rules. Data expressed in the constructs of the target format will be encoded into target data files according to the encoding rules defined by the target format.

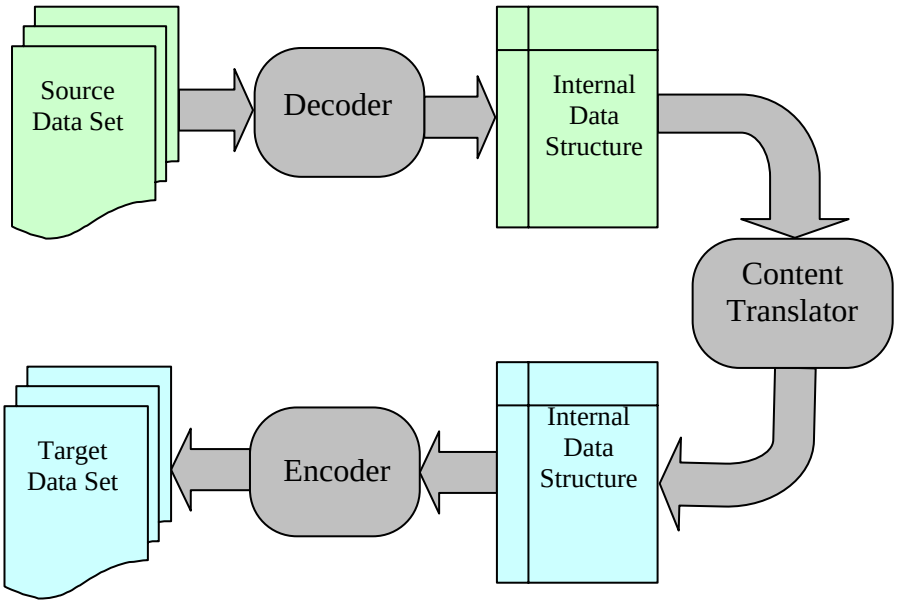


Figure 3.1 Structure of mechanical geodata translators

It is now possible to analyse how data organization differences are resolved in this translation process. The decoder reads and interprets data encoded in files to some form of internal data structure that is to be processed by the content translator. The content translator converts the internal data structure to another internal data structure that can be understood by the encoder of the target format. What actually is done is a conversion of the data schema in the source data model into a data schema in the target model. Therefore, the physical file format difference is resolved by turning data to or from an intermediate (often in-memory) physical structure by means of decoding and encoding. Differences in logical data models are resolved by converting internal data structures. The problems with mechanical translators now follow.

3.2.2 Lack of Support to Semantic Reconciliation

First, the data schema translation is hard-coded. Although most GIS products adopt the so-called geo-relational data model, the object-oriented model is the trend and

there have been several geodata exchange formats using the object-oriented data model (SAIF 1995, OGC 2002). When the model conversion is between two variants of the basic geo-relational model, the conversion is quite straightforward and it neither gives much room nor raises much need of customization. However, when the model conversion is from an object-oriented model to a geo-relational model, it is very likely that light or heavy customization is very much desired, although a default conversion rule will work in some cases. The situation is similar when converting the conceptual database design in the E-R diagram form to a relational data schema. In the reverse case, when the model conversion is from the semantics-poor geo-relational model to the semantics-rich object-oriented model, the translation operator may wish to make some implicit semantics explicit in the resulting object-oriented data schema. Given a data schema in the source data model, a hard-coded model conversion generates a fixed data schema in the target data model in the way the translator developer thinks to be best. It does not allow user intervention.

Second, the spatial data type conversion is hard-coded. As is known, a spatial data model is to a large extent characterized by the spatial data types it supports. In geodata translation, the logical data model conversion consists of two parts, the data schema conversion (as discussed above) and the spatial data type conversion. Each GIS, CAD, or other graphics-related software tool supports a limited number of spatial or geometric data types. That will often lead to limitations in application modelling. For example, if the source format does not support the 3-D point data type while height information needs to be represented, the user will employ an ordinary height attribute together with a 2-D point type attribute instead of a preferred 3-D point type attribute. When translating data into another data format that does support the 3-D point type, the user would normally want to assemble the two

attributes into one 3-D point attribute. The point is that there is an interaction between the fixed spatial schema and custom application schemas. Such interaction leads to the need for customizing the application schema so that it can be best represented in a specific spatial data model. Hard-coded model conversion excludes desirable customization.

Third, when there is a difference between the application's conceptual model and the underlying conceptual model of the data being translated, it is likely that a translation operator would want to tailor or define the translation by taking into account semantics not captured in the data schema but specified in documents or obtained by other means so that the resulting data is consistent with or closer to the target application's semantics. For instance, the target application may only be interested in the roads that are with four or more lanes while the source data contains roads with any number of lanes. One natural way to do that tailoring would be similar to the way *views* were defined in the database systems on the basis of *tables*. That is another reason why one would want to customize the data schema resulting from an automatic model conversion process.

In summary, the mechanical translator works only on formal semantics of data models. It does not take into account the semantics of a specific dataset and the requirements of a particular application. Mechanical translators do not provide a method that allows the translation to be customized and thus become smart or understanding.

3.2.3 Poor Reusability and Lack of Coordination in Translator Development

The other problem of mechanical translators is related to their implementation. It is in fact a problem not inherently associated with mechanical translators but with the way they have been implemented. The model shown in Figure 3.1 is widely used in implementing geodata translators and is believed to be able to facilitate writing data translators (Lazar 1992, GDBC 1994, Altheide 1996). This practice will now be examined closely. To help make standards widely used, it is desired that programs that can read data from and write data to the specified format are available to developers of data translators. With the model of the mechanical translator in mind, some standardization efforts have developed data decoding and encoding program libraries. For example, sdts++ is a program library provided by USGS (United States Geological Survey) to the public to help develop data translators (Altheide 1996). Similarly, the SAIF format also provides data accessing API (Geographic Data BC 1994). There have also been a few open source libraries on the web for some popular vendors' data formats such as MIF, SHAPE and DGN formats. Providers of these libraries tend to think that implementing geodata translators between a pair of formats would be easy if the decoding/encoding programs of the two formats are available. This seems quite good. In practice, however, using geodata format APIs is not trivial. The following factors contribute to making such practice far from pain free.

First, the APIs are hard to use. Geodata format APIs often come as source code libraries. Although most GIS software tools have evolved into component-based software architecture and provide highly commercialized software components for application development, no vendors provide data manipulation libraries for their

data formats, which are integral components of their products, for free. What can be available is at most a minimum format accessing API source code with minimum documentation, rather than a set of highly commercialized components. It is always necessary to look into the internal implementation to make use of them. They are provided as they are. User may need to fix any problems found with the translator or to make enhancements. Even if the library is a complete one, subtle changes may be needed to have them work on a platform other than the developing platform. The code may be in a language different from the developer's favourite. The two libraries for the source and target formats are likely to be in different programming languages, say one in C and the other in Java.

Furthermore, the APIs use low-level data abstraction. Due to the lack of coordination of translator development among vendors, users and data producers, each library uses its own internal data structure. Each API works with only one format. API users need to get very familiar with as many low-level APIs as the data formats he or she needs to handle. This is similar to the situation of accessing databases or legacy datasets from programming languages before standard APIs like ODBC or JDBC come into play. The internal data structures provide diverse low-level data abstractions. No common high-level data model is applied. They do not or cannot support high-level data manipulation. What is provided is minimal format accessing function, yet it is hard to use.

With the above discussion, it can be realized that the content translator built on the basis of these decoding/encoding program libraries will produce similar problems. With no common data model and common spatial schema, and being built on top of diverse internal data structures, components or modules of a content translator

become hardly reusable for developing other content translators and content translators duplicate the functions of each other repetitively (Lee 1990, 1997)

In conclusion, developing geodata translators according to the model shown in Figure 3.1 provides poor component reusability.

3.3 Semantic Translation

3.3.1 Semantic Reconciliation: Facilitating versus Automating

As explained in Chapter 2, semantic heterogeneity needs to be resolved to make proper use of data obtained externally. It is desirable that semantic geodata translation can be automated in a manner similar to that of conventional data translation, which automatically converts data from one format to another. Logically, the extent to which semantic reconciliation can be automated depends on the extent to which semantics can be formalized and computed. Full automation of semantic reconciliation requires full formalization of semantics, automatic detection of semantic heterogeneity and automatic resolution of semantic heterogeneity.

Efforts have been initiated to formalize data semantics. Ontology in the AI (artificial intelligence) field is the technology developed to formalize conceptualizations and to reason the semantic relationship between concepts (Gruber 1993, Guarino 1998, Chandrasekaran 1999, Kuhn 2002, Frank 2003, Bowers and Ludäscher 2004). It is also the technology on which the WWW consortium is basing the ambitious vision of Semantic Web (Antoniou and Harmelen 2004). There has also been research on formalizing contexts and performing context mediation (Goh et al. 1994, Reddy and Gupta 1995, Akman and Surav 1996, Goh 1999). All these technologies are related to formalizing semantics and enabling semantics computing and therefore worth

keeping an eye on. On the one hand, it seems reasonable to believe that semantic reconciliation can be progressively automated in the long run. On the other hand, it should also be noted that full formalization of data semantics is still very much in its infant stage and current data management technology knows little of integrating AI research results (Lee et al. 1996). Many researchers believe that full automation of semantic reconciliation in an open context is not feasible given the-state-of-the-art of theory and technology of computing semantics (Sheth and Larson 1990, Heiler 1995, Colomb 1997). One of the arguments concerning automatic schema integration (Colomb 1997) is cited below:

One reason why a completely automatic schema integration process (particularly for discovering attribute relationships) is not possible is because it would require that all of the semantics of the schema be completely specified. This is not possible because, among other reasons, (1) the current semantic (or other) data models are unable to capture a real-world state completely (2) it will be necessary to capture much more information than is typically captured in a schema, and (3) there can be multiple views and interpretations of a real-world state; and the interpretations change with time (Colomb 1997).

Based on the above observations, this study takes a practical approach to dealing with semantic issues in geodata translation. This study chooses to facilitate, rather than to automate, semantic reconciliation. That is, mechanisms and tools will be devised to help human operators resolve semantic heterogeneity and make their task easier. In particular, it is assumed that human operators will identify semantic heterogeneity and use the devised high level language to specify semantic reconciliation. This approach of high level specification is widely adopted in data translation (Bailey 1995, Denno 1999, Lee and Xu 1999, Xu et al. 2000, Safe Software 2005b and 2005c) and database integration (Motro 1987, Chomicki and

Litwin 1993, Hurson et al. 1994, Kelley et al. 1995, Navathe and Savasere 1996, Gingras et al. 1997, Devogele et al. 1998, Parent and Spaccapietra 1998, Elmagarmid et al. 1999, Pottinger and Bernstein 2003, Halevy et al. 2005) to achieve semantic interoperability. The next sub-section examines the FME™ geodata translation system, which takes such an approach.

3.3.2 FME™ as a Semantic Translator

Perhaps the most significant achievement in geodata translation technology is what is realized by Safe Software (Safe Software 2005a) in their geodata translation product, FME™ (Feature Manipulation Engine). FME™ is designed to be a translation system dealing with many formats rather than a single translator between two specific formats. As such, FME introduces an internal intermediate data representation that supports sophisticated spatial processing. That is why it gets its name, feature manipulation engine. Logically, to translate data from one format to another FME first converts data in the source format into the internal representation and then does the required processing and finally converts data from the internal representation into the target format. As a system by itself, the intermediate representation is to some extent transparent to users, no matter how complicated or simple it is. This characteristic is in contrast to using a standard format as the intermediate. The spatial processing capability built in FME is reusable when dealing with any specific pair of formats through modular software design, which facilitates the writing of the translator between any specific pair of formats. So, to FME users, there is a direct translator from one format to another as long as they are among those supported. This characteristic is again in contrast to using a standard format as the intermediate. FME also supports semantic translation (Safe Software 2005b, 2005c),

i.e. changing the view of data. This can be done by developing a custom mapping file with the support of a large library of FME functions and factories. Unfortunately, such customization is non-trivial. In fact, it is a quite confusing and involved task. This is largely due to the fact that FME lacks formal definition of each data format and is based on proprietary technology. For instance, a) no universal schema definition language is used to define each data set, and b) the scripting language is a proprietary procedural language. In this study, a more formal approach is taken and the use of open and standard information technologies to facilitate writing geodata translators and support semantic translation is investigated.

3.4 Geodata Translation and Database Systems

3.4.1 Data Translation in Pre-RDBMS Time

In the 1970s, data translation was studied in the database (the term *databank* was used more often than *database* at that time) field as a problem encountered when migrating data from one system to another or when applications evolved from one design to another (Navathe and Fry 1976). At the time, database systems had just evolved from file systems. Data formats were hardware dependent and divergent data models were employed (Fry et al.1972). Data translation was carried out by manual programming for each pair of source dataset and target dataset. To overcome the difficulty of data translation, many generalized data translation models (Fry et al.1972, Shamkant et al. 1976, Shu et al.1977) were proposed and prototype systems implemented. These studies agreed that the problem of data translation was largely due to the lack of explicit data definition and the lack of a high-level data manipulation language (Fry et al.1972). An important study is the EXPRESS system (Shu et al. 1977) developed to "*test the generality and applicability of high level data*

description and high level data manipulation techniques to the data translation problem". EXPRESS uses two non-procedural languages, DEFINE and CONVERT, to specify three modules that are then automatically generated: a reader, a restructurer and a writer. The so-called high-level data description language corresponds to a combination of a file structure definition language and a data definition language, while the so-called high-level data manipulation language corresponds to something like SQL.

With the maturing of relational database theory and the dominance of standardized relational DBMS, data translation is no longer a significant problem in the data transfer between relational database systems.

3.4.2 Database System as a Data Manipulation System

As the use of database-like technology is advocated for the purpose of geodata translation later in this study, it is worth recalling that database technology was invented to facilitate data storing, accessing and manipulation. In the early days of developing computer applications, developers used the file I/O (input/output) mechanism provided by programming languages to implement data persistence by themselves. Although it is ideal that an application mimic the database approach by designing logical and physical data models and implementing data encoding and decoding rules, implementing the data structure abstraction was non-trivial. In addition, database theory and technology were not sufficiently mature at that time. Applications usually chose the shortcut approach of hard coding data structure and data accessing, known as the data file approach. The following disadvantages have been observed regarding this approach (Ullman and Widom 1997, Elmasri and Navathe 2000):

- The application-specific data structure implemented is hard coded within application programs. That is known as the program-data dependence. A change in logical data structure requires changes in the physical data structure, which in turn results in changes in the application programs. Maintaining consistency among physical data structure, logical data structure and application programs is error-prone, tedious and time-consuming.
- Data accessing and manipulation is carried out in a procedural way. To access data, several routine steps are needed including a) defining in-memory data structure, b) allocating memory space, c) locating data in external storage interactively or according to some application-dependent conventions, and d) reading data from an external file and putting them into the memory according to rules specific to each data structure. For data manipulation, the routine steps include a) defining the in-memory data structure for the resulting data, b) allocating space for resulting data, and c) restructuring data according to rules specific to each manipulation. To make data persistent, the routine steps include a) creating one file or more files to hold data, b) writing data out to the file according to encoding rules specific to each data structure, and c) closure of the file(s).
- Each application implements its own specific data structures without common logical-level and physical-level models. When other applications want to reuse the data for their own purposes, they must understand the logical structure of the data and also know the physical structure of the data. That introduces much difficulty in data sharing among related applications.

Database systems facilitate data storing and data accessing in individual applications. With the physical structure transparency provided by DBMS, interactive data

manipulation is free of physical structure details and data manipulation programming is greatly simplified. Such convenience is achieved through the data structure abstraction mechanism. Remember that a data schema is constructed from the constructs provided by the underlying meta-data model. The DBMS converts declarative schema-based data manipulation statements into procedural physical structure-based instructions. For a relational database, the famous declarative SQL language greatly facilitates data manipulation.

3.4.3 Geodata Translation based on Database Systems

An important idea of using database technology for geodata translation was proposed by Pascoe and Penny (1990). In their study, data exchange was viewed as letting data pass through a series of interfaces, namely format decoder, translator and format encoder as shown in Figure 3.2. According to Pascoe and Penny (1990), relational DBMS is particularly useful for the translation phase of data exchange:

> *With a relational database, the whole of the source data can be set up as a small number of input relations. In the translation phase, data in the source relational model are converted into the target relational model, using (for the most part) expressions in terms of input relations and relational operators. Since relational operations are specified at a high level, the effort for the "programmer" of the interface is greatly reduced. (Pascoe and Penny 1990)*

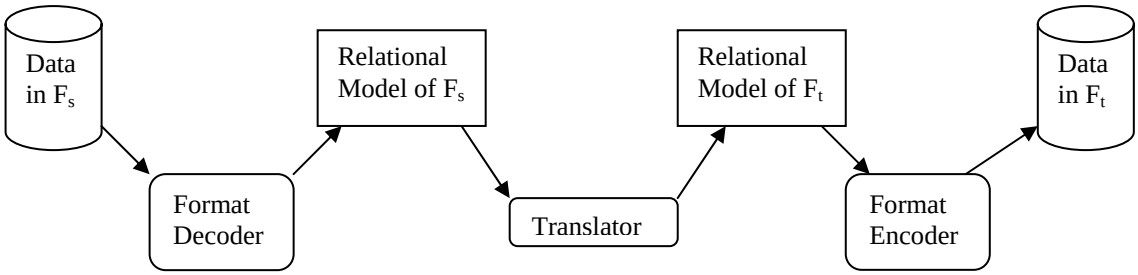


Figure 3.2 A model of geodata translation using relational database systems (adapted from Pascoe and Penny 1990)

No serious study has been carried out to explore that potential due to the fact that it is widely believed RDBMS is not able to deal with the complexity of geodata effectively. However, since the 1990s, great changes have taken place in the database field. Nowadays, there are object-oriented database products (Bertino and Martino 1993, Barry 1996, Cattell and Barry 1997, Loomis and Chaudhri 1998, Cattell et al 2000). Object-oriented features have also been introduced into relational databases, resulting in the so-called object-relational DBMS (Carey et al. 1997, Oracle 2001). Moreover, DBMS from major database vendors are providing support to spatial data (IBM 2004, Oracle 2005). Further, XML database technology is emerging (Chaudhri et al. 2003, Zohra Bellahsène et al. 2004). Application of these new database technologies to semantic geodata translation deserves new studies. It is argued here that the database-based approach to geodata translation is especially useful a) for constructing large scale data centres that integrate data from various sources and distribute data to users according to their respective requirements and b) for building dedicated geodata translation systems that support direct translation among a large number of formats, such as FME^{TM}. However, database-based geodata translation in

the public domain will not be a good idea as it is inherently DBMS-dependant and not sufficiently flexible for occasional use.

3.5 Data Exchange in other Disciplines

In areas where relational DBMS failed to fulfil application requirements, data exchange is a serious problem as already noted in the GIS field. Two of the areas are electronic publishing and CA*x*, i.e. computer-aided *something* such as *design* and *drafting*. Unlike the situation in the GIS field, these two areas pay more attention to the development of sophisticated data exchange techniques, as well as to standardization.

The SGML (Standard Generalized Markup Language) technology (Smith 1992), from which XML technology is derived, is developed for exchanging electronic publishing data. As mentioned earlier, this study will explore using the XML technology for geodata exchange, so we do not elaborate on SGML here. Suffice to say that SGML is successful but it is also known as an expensive technology, adopted mainly by large enterprises (Goldfarb and Prescod 2002). While XML aims at facilitating data exchange, it bears much similarity to database technology.

The CA*x* field has developed the STEP (STandard for the Exchange of Product data) (Fowler 1995) technology. Sophisticated data translation techniques have been developed, including the semantics-rich data definition language, EXPRESS (ISO 10303-11:1994), high-level API (ISO 10303-22:1994) and mapping languages such as EXPRESS-M (Bailey 1995) and EXPRESS-X (Denno 1999). Moreover, its implementation can be built on top of database systems for processing efficiency and for tackling large datasets (David 1998).

3.6 Summary

The extensive standardization efforts on geodata transfer have not achieved the expected success, while geodata translation techniques are underdeveloped. It is argued that given the complexity of the geodata sharing problem, both standards and sophisticated data translation techniques are needed to facilitate geodata exchange, of which the latter is the emphasis of this study.

Mechanical geodata translators cannot be customized and thus cannot allow reconciliation of semantic differences. The API-based approach to facilitate writing geodata translator is of very limited use.

While FME™ can be considered as a semantic geodata translation system, it lacks conceptual clarity and is difficult to use. As a proprietary technology, no deep insight can be gained into it. Better and public domain technology is highly desired.

Data exchange techniques developed in other disciplines provide valuable experiences for geodata exchange. It can be said that the overall trend is to develop or apply sophisticated data manipulation technology similar to database systems to facilitate data translation. That is what was done in the pre-RDBMS time and also the approach taken in SGML and STEP technologies.

Chapter 4

A Model-Driven Framework for Geodata Exchange

In Chapters 2 and 3, the problem of geodata translation has been stated and limitations of current techniques analysed in detailed. In this chapter, a model-driven framework for geodata exchange will be proposed. The aim of the framework is to enable automatic generation of geodata translators and to introduce a semantic reconciliation mechanism into the geodata translation process. This chapter is organized as follows. The 3-layer model of within-system data representation is first extended to a 4-layer model of data representation for addressing inter-format issues in Section 4.1. Then, a 3-layer model of geodata translators is formulated in Section 4.2. In Section 4.3, the relationship between translation specification and translator generator is first outlined and then a mode-driven method of generating geodata translators is proposed. Section 4.4 discusses how the methodology in conjunction with suitable technology can serve as a framework for geodata exchange. Section 4.5 explains the relationship between the proposed methodology and framework and OMG MDA initiative. Finally, Section 4.6 summarizes this chapter.

4.1 A 4-Layer Model of Data Representation

As shown in Chapter 2, data is organized in two levels of abstraction within a database system or an exchange format, namely the logical level and the physical level. The two levels are bridged by the DBMS or the software tool that implements the encoding rules. In addition, an implicit logical data model or meta-formalism is implemented by the software and used to specify data schemas. That model of data

representation can be generalized as the 3-layer model of data representation as shown in Figure 4.1. The three layers in the model are the *data instance*, *data schema* and *meta-formalism* layers. In the 3-layer model, the meta-formalism is a model of all data schemas while a data schema is in turn a model of some data instances. It should be emphasized that the three layers are tied together by associated software. Within a system, there is a common (but proprietary) language, i.e. the meta-formalism, for specifying data schemas and a common (but proprietary) file format for encoding instance data.

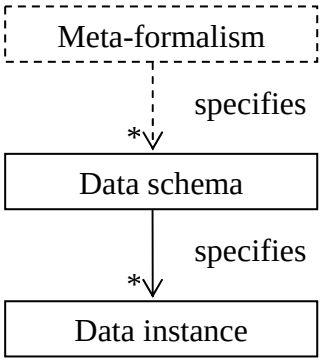


Figure 4.1 A 3-layer model of data representation of a data format

This is all right if data are kept within a system. When data goes out of the system, the logical data model or meta-formalism used to specify data schemas needs to be specified to be compared with that of a target system. A formally specified logical data model is called a *meta-schema* here. A meta-schema is obtained by explicating a logical data model of a software package or an exchange format using some meta-meta formalism as illustrated in Figure 4.2. That results in a 4-layer model of data representation as shown in Figure 4.3. Note that in the model, the same meta-meta-formalism is used to specify meta-schemas of various data formats. Suppose that in the 4-layer model, there is also common syntax for representing data schemas and instance data. Then, we can obtain a counterpart of a proprietary data format in the 4-

layer model by defining rules of converting the proprietary syntax of encoding instance data and data schema (or meta-data) into the common syntax. Thereby, the 4-layer model is in principle able to accommodate as many data formats as there may be.

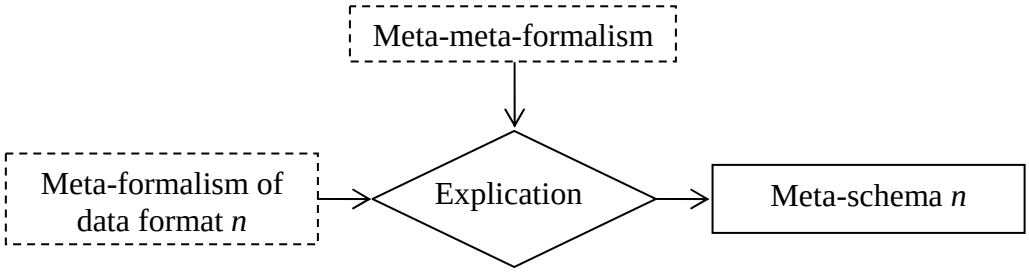


Figure 4.2 Explicating meta-formalism of a data format

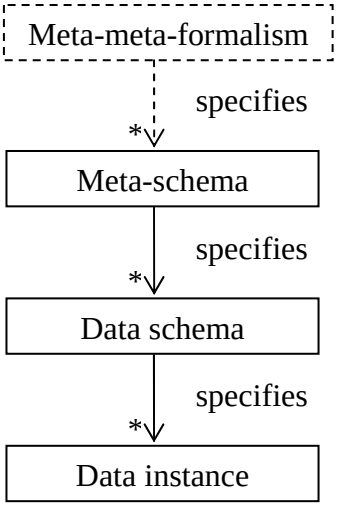


Figure 4.3 A 4-layer model of data representation of the framework

4.2 Layers of Translators

In Chapter 3, the structure of a conventional geodata translator was analysed. According to the analysis, a conventional geodata translator resolves differences in the three layers of data representation using a decoder/encoder and content translator.

That is, a geodata translator consists of two layers as illustrated in Figure 4.5 (a). It is a good idea to enable the reuse of code. However, due to lack of interoperability among programming languages and the lack of high quality API in the public domain for various data formats, the potential benefits of developing translators by assembling layered components are not realized.

In this study, the idea of layering geodata translators will be retained and extended. First, in order to make the benefits of layering geodata translators realized in practice, a common syntax for data (and meta-data) representation will be introduced to decouple the tight integration of layers of geodata translators. That is, a decoder/encoder will no longer convert persistent data in a proprietary format into some proprietary internal (often in-memory, but not always) data representation for later processing. Instead, they convert data from/to a proprietary persistent format to/from a common persistent format as shown in Figure 4.4. In that way, proprietary decoding/encoding APIs are decoupled from the remaining parts of a geodata translator and thus details of those proprietary APIs are no longer of concern.

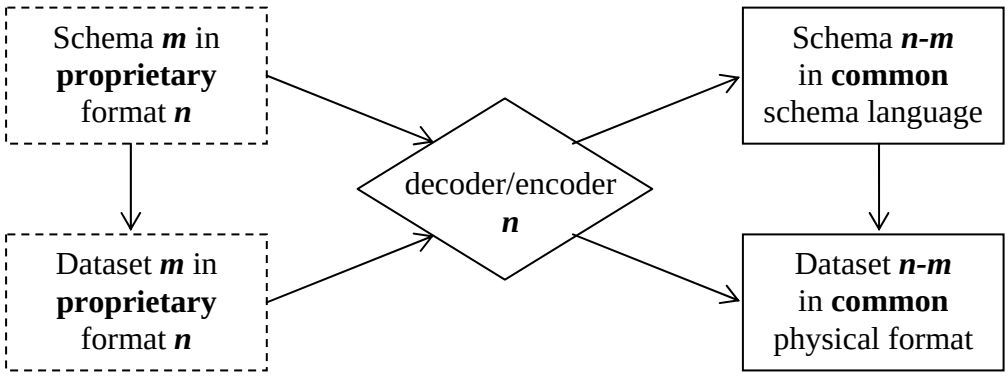
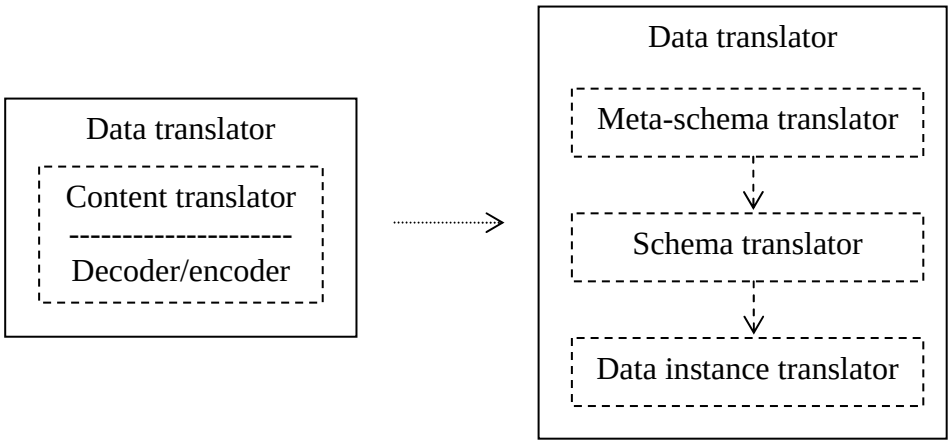


Figure 4.4 A decoder/encoder converts an instance dataset and its associated schema between a proprietary format and its counterpart in the framework

In addition, as can be seen, there is a mismatch between the 2-level layering of translators and the 3-level layering of data representation. That is because a specific geodata translator for a specific pair of source and target geodata formats takes into account only the two involved formats and it hard codes the rules of meta-formalism conversion. Moreover, by hard coding meta-formalism conversion, it blocks us from customizing the transformation of a specific source data schema to an intended target data schema.

To enable the reuse of data translator components and customization of data translation, the content translator of a traditional geodata translator needs to be divided into two layers, namely the schema translator and the meta-schema translator. The idea is illustrated in Figure 4.5(b). In the layered model of data translators, a data translator consists of three layered components, namely a meta-schema translator, a schema translator, and a data instance translator. As has been shown, components in a traditional data translator are tightly integrated and translation rules are hard coded. In the proposed layered model of data translators, the components will largely be generated and customizable as explained in the next section.



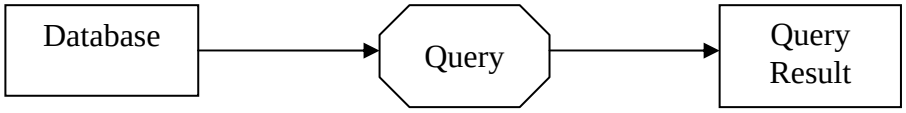
(a) Traditional data translator (b) 3-Layer data translator

Figure 4.5 Layered models of data translators

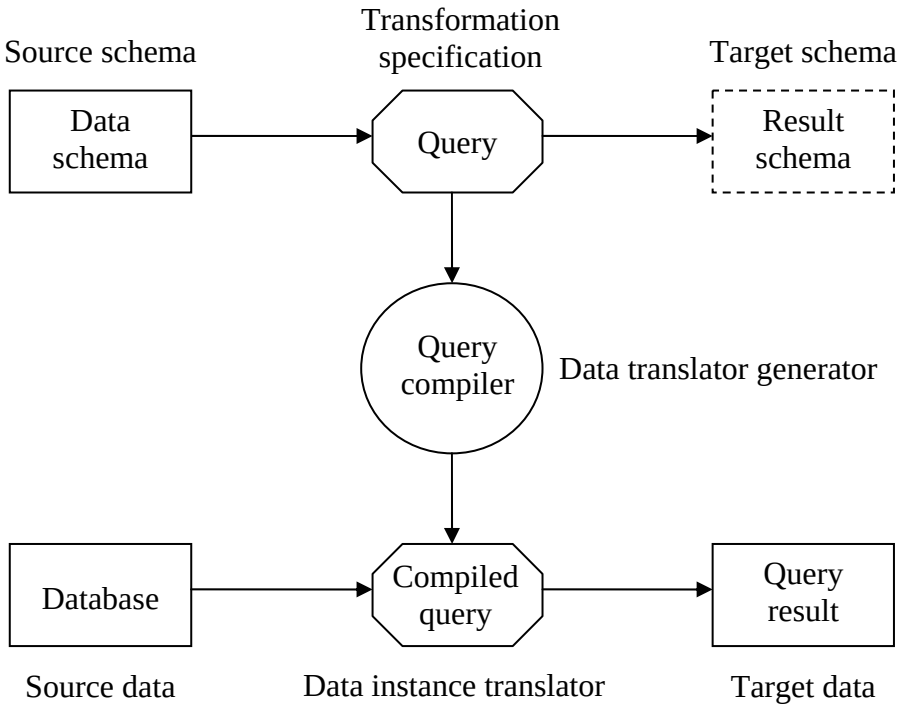
4.3 A Model-Driven Method of Generating Geodata Translators

4.3.1 Translator Generator and Translation Specification

From an abstract point of view, a data translator is a program implementing a kind of transformation that transforms input data into output data. If the program accepts "parameters" to allow customizing the transformation process to some extent, it is regarded as customizable. If it accepts full customization of the transformation process, it is in fact a translator generator that accepts transformation specifications. That is in contrast to mechanical translators, which athough producing different output data from different input data have transformation rules hard-coded and never accept transformation specifications. In that sense, a database system is a translator generator. It accepts queries, which actually are transformation specifications, and produce query results, i.e. output data, from a database, i.e. input data. This view is illustrated in Figure 4.6. However, a database system accepts only queries written in its provided query language and transforms only data represented in its internal format. If it is to be used as the basis for implementing a generic data translator, it needs to be extended to accommodate diverse data formats. That possibility will be explored later in this chapter.



(a) The simple view of database querying



(b) A data translation view of database query processing

Figure 4.6 Database systems as data translator generators

Regarding a database system as a data translator generator shows that a data instance translator can be generated from a schema-level transformation specification. Similarly, we can imagine that a data schema translator may be generated from a transformation specification at the meta-schema level. That idea leads to model-driven data translation.

4.3.2 Generating Lower Level Translators from Upper Level Specifications

Suppose that there are methods or languages for specifying transformations or mappings at schema and meta-schema layers of the formulated 4-layer model of data

representation. Suppose also that there is a mapping processor that generates translators from mappings. Then, data instance translators and schema translators can be generated automatically from schema mappings and meta-schema mappings, respectively. Further, suppose that an upper-level mapping can be used to derive a default lower level mapping as illustrated in Figure 4.7.

With the above assumptions, a data instance translator can be generated and customized as illustrated in Figure 4.8. In Figure 4.8, the dotted line indicates how the translation process proceeds. Once a meta-schema mapping from a source format to a target format is provided, it can be used to generate the schema translator and the schema mapping generator for the pair of formats. The schema translator accepts a source schema and translates it to a default target schema. At the same time, the schema mapping generator produces the associated schema mapping between the source and target schemas. If needed, the target schema and the associated schema mapping from its source can be customized. The schema mapping can then be used to derive a data instance translator that translates source data into target data that conforms to the target schema.

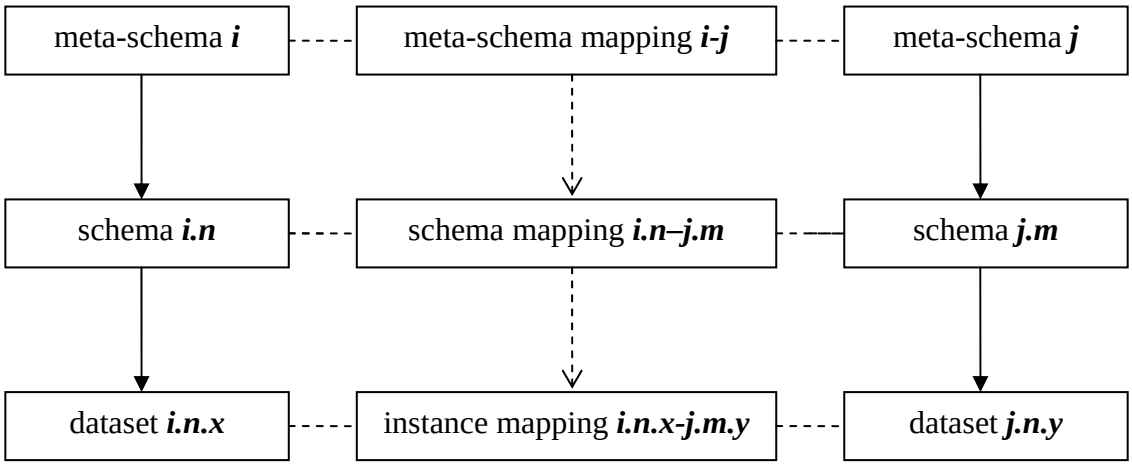


Figure 4.7 Chaining model transformations (or mapping) at different layers

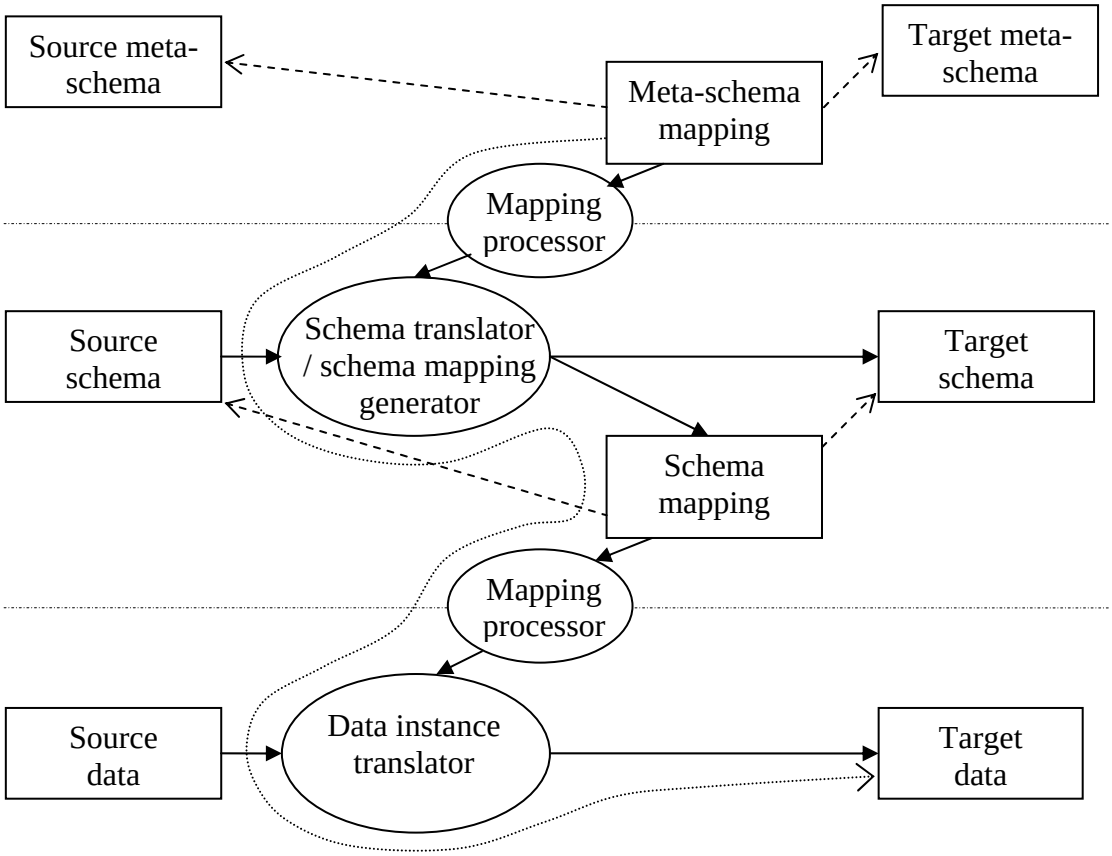


Figure 4.8 An illustration of model-driven geodata translation

Up to now, a model-driven methodology of geodata translation has been conceptualized. As can be seen, data translation has been formulated as a top-down chain of transformations, starting from the meta-schema level, via the schema level and down to the instance level. As a transformation specification at the upper level actually models and is used to derive the transformations at the lower level, the method is thus considered *model-driven*. The benefits of the method can be seen clearly if it is put into the context of geodata exchange practice as analysed in the following section.

4.4 Toward a Model-Driven Framework for Geodata Exchange

The proposed model-driven method of geodata translation, together with the 4-layer model of data representation and the 3-layer of geodata translators, constitutes a methodology for geodata exchange and geodata translation. This methodology, in conjunction with suitable technology, can form a model-driven framework for coordinating geodata exchange practice in a flexible, highly automated and highly customizable way.

To complete the framework, formalisms for data representation at each layer need to be realized as well as formalisms for specifying mappings at meta-schema and schema levels. Most importantly, these formalisms need to be public domain technology instead of proprietary technologies. Two candidates are MDA (Model-Driven Architecture) technology (Miller and Mukerji 2003) and XML technology. While MDA is more oriented to software engineering, XML technology was developed to facilitate data exchange. In the next three chapters, the use of XML technology as the basis for implementing model-driven geodata translation will be explored in depth. For the moment, let us assume the required technology is available and examine how the framework can coordinate geodata exchange practice.

Accommodating diverse geodata formats

An existing proprietary data format can be incorporated into the framework by defining a counterpart for it in the framework (assuming the format is open to the public) and developing the associated decoder/encoder. Formats of new software tools can directly use the syntax of the framework for encoding instance data and data schemas and to specify its meta-schema using the meta-meta-formalism of the

framework. Hence, various geodata formats of legacy, current and forthcoming systems can be naturally accommodated in the framework without significant restrictions.

Format-to-format translation

With two data formats specified, one (or more) meta-schema mappings from one to the other can be specified. From the meta-schema mapping, the bulk of a translator can be generated automatically. Moreover, at the format or system level, translation can be customized through specifying and using alternative meta-schema mappings.

As data are to be translated directly from one format to another, information loss can be kept to a minimum. In addition, the complexity of customizing translation is only related to the complexity of the two formats involved and the problem at hand. No extra complexity is caused by the introduction of complex intermediate geodata formats.

Dataset-to-dataset translation

With meta-schema mapping specified, default schema mapping for a specific dataset can be derived automatically and used as a basis for dataset level customization. Agile geodata translation can be performed on a dataset-to-dataset basis, even without defining format-to-format mappings. In such cases, a schema mapping can be specified for the dataset to be translated, probably with supporting functions. A data instance translator can then be generated specially for the dataset. For example, a transportation application may only be interested in the road layer of a base map provided by some mapping agency, who adopts its own data format for the rich set of data it provides. The data user does not want to deal with the full complexity of the

data provider's format and simply want to make use of the road dataset. In this case, a direct dataset-to-dataset translator is enough and simpler to create, use and maintain than a format-to-format translator.

4.5 Comparison and Relation to Other Efforts

4.5.1 Comparison with the Brute-force Programming Method

Traditionally, geodata translators have been developed by brute-force programming. The model-driven method proposed is a method for automating the development of geodata translators. The automation is based on the conceptual basis of the 4-layer model of data representation and the 3-layer model of data translators. With the method, components of a data translator at the schema and instance layers can be automatically generated. The component at the meta-schema layer is specified using the assumed high-level meta-schema mapping language. Therefore, the work in developing a geodata translator from one format to another is greatly reduced.

Mechanical translators developed through brute-force programming do not allow customization of the geodata translation process. In contrast, with the model-driven method, the geodata translation process can be customized easily through high-level specification. Mapping specification at the meta-schema layer enables customization of geodata translation on the format-to-format level, while mapping specification at the schema layer enables customization of geodata translation at the dataset level. Therefore, semantic translation is facilitated through translation customization.

4.5.2 Comparison with Standard-based Geodata Exchange

The rationale of facilitating geodata exchange through the adoption of a geodata standard is the use of a universal language for geodata exchange, which thus dispels differences in data organization or data semantics. In contrast, the proposed model-driven framework facilitates geodata exchange through automating the generation of geodata translators that resolve these differences. Although the two approaches seem contradictory in principle, they are largely complementary in practice. That is because there cannot be an invariable geodata exchange standard that is applicable to all application fields. To facilitate geodata exchange among legacy and current systems and among diverse application fields, automating the development of geodata translators is a desirable complement to the standardization approach.

4.5.3 Relation to the OMG MDA Initiative

The proposed methodology for geodata translation and framework of geodata exchange has been inspired by the OMG (Object Management Group) MDA (Model-Driven Architecture) initiative.

The OMG MDA initiative aims to allow the definition of computer readable applications and data models that allow long-term flexibility of implementation, integration, maintenance, and testing and simulation (Miller and Mukerji 2003). The philosophy underlying MDA is that the diversity in implementation technologies should be hidden from conceptual design and specification. That is similar to the old idea that physical data storage and the accessing of details should be hidden from logical level data definition and manipulation in database systems, or programming should be done using high-level programming languages instead of machine

languages. So, it is not a new idea. It is nevertheless important because it is in essence advocating building a new higher level of abstraction on top of current high-level computer languages; in other words, further raising the level of abstraction (Poole 2001, Mellor et al. 2004).

Currently, the MDA framework has defined the Meta Object Facility (MOF) for meta-modelling (OMG 2001b), XML Meta-data Interchange (XMI) (OMG 2001a, 2003b) for exchanging models and meta-models, and the Common Warehouse Meta-model (CWM) for data warehousing environments (OMG 2003a). These form the core of MDA technology. The bottleneck of MDA technology is the lack of automatic model transformation technology (Gerber et al. 2002). Intensive research efforts are ongoing to conquer this difficulty, and many consider that model transformation can be automated by means of graph transformation (Andries et al. 1999, Agrawal et al. 2002) or variants of production rules (Revault 2003). Although current MDA technology is not readily usable for geodata exchange, its basic ideas are applicable and have been incorporated into this study; namely, raising the level of abstraction and having models drive software automation or development. The term *model-driven* instead of *MDA-based* has been chosen to describe the methodology proposed in this study because MDA technology has not been chosen as the basis for implementing geodata translation in this study.

4.6 Summary

In this chapter, a model-driven methodology of geodata translation has been formulated. It consists of a 4-layer model of data representation, a 3-layer model of geodata translators and a model-driven method of generating geodata translators. The 4-layer model of data representation enables explicit specification of meta-schemas

of diverse geodata formats. Common formalisms are introduced (assumed) at each layer of the model to provide a basis for a common method of data (or meta-data) manipulation. The 3-layer model of geodata translators reveals the layering of components of geodata translators. It provides a conceptual basis for the automatic generation of geodata translators. The method for automatically generating geodata translators was inspired by the OMG MDA initiative. The notions of meta-schema mapping and schema mapping were introduced. Then, mapping processors were assumed to be available. Based on that assumption, geodata translation was conceptualized as a chained model transformation process in which upper level mappings are used to derive lower level mappings and translators. Hence, the methodology enables automatic generation of translators as well as translation customization at both meta-schema and schema levels. On that basis , a model-driven framework for geodata exchange came into sight. The framework can accommodate diverse geodata formats and facilitate both directly format-to-format translation and agile dataset-to-dataset translation.

The focus of the following three chapters will be on exploring XML technology as a technical basis of the proposed framework and developing new techniques as required. Issues addressed include formalisms at each layer of data representation, methods for specifying (meta-) schema mappings and associated algorithms, and the method and algorithms for chaining meta-schema mappings to schema mappings. Prototype implementation and test cases are also presented.

The proposed framework can also be used as the reference architecture to build proprietary geodata translation systems as part of a geodata clearinghouse or online geodata translation services. Two published papers co-authored by this author

discussed the design of such systems (Lee and Xu 1999) and the use of the schema mapping method for semantic reconciliation (Xu et al. 2000), respectively. When building proprietary geodata translation systems, current spatial database systems provide a good basis for implementation. As the main objective of this study is to develop geodata translation technology suitable for public domain use, the problem of building proprietary geodata translation systems will not be elaborated in the rest of this study.

Chapter 5

XML-Based Geodata Exchange

The framework of geodata exchange presented in the last chapter needs associated down-to-earth technology for it to come into being. This chapter explores the XML (Extensible Markup Language) technology as the technological base. The purpose is to examine how XML fits in (if at all), to discover potential blanks or gaps and to identify if any missing blocks and joints need to be developed.

XML emerged as a simple file format for document and data exchange on the Web. Since the release of W3C XML 1.0 Recommendation (W3C 1998) in early 1998, a rich set of technologies has been developed on the basis of this simple syntax and form the group of XML technologies for data modelling, storage, exchange, management and retrieval, etc. XML is becoming the standard way of exchanging data between software applications (Harold and Means 2002). Many database vendors are providing XML support in their products (Microsoft 2001, Wong 2003, Oracle 2005b). This is because XML provides unmatchable flexibility and power for data exchange.

This chapter first gives an introduction to XML technology in Section 5.1. Then, the relationship between XML technology and the proposed framework for geodata exchange is analysed in Section 5.2. This is followed by a comparison of 2-layer and 3-layer models of XML-based data formats in Section 5.3. In Section 5.4, methods for developing 3-layer XML-based (geo-) data formats are discussed and these methods are applied to develop XML counterparts of SHAPE and MIF formats. The benefits of using XML techniques to develop mechanical translators are analysed in

Section 5.5. Section 5.6 gives an analysis of using XSLT together with supplementary imperative programming for reconciling structural and semantic differences between source and target data. Finally, a summary of the exploration is given in Section 5.7.

5.1 XML and its Associated Technology

As is known, data exchange is a basic communication means between loosely coupled systems or sub-systems of a large system. Most of the numerous data formats devised for data exchange are designed to work for a specific field of application and not extendable to accommodate new applications. A universal data exchange format or mechanism that can serve a wide range of applications is very much desired not only for the geospatial information field but also for other fields, such as electronic design data, electronic business messages and electronic documents for publishing and so on.

There have been two ISO file exchange standards that have the potential to serve as a universal data exchange format, namely ISO 8879 (SGML, Standard Generalized Markup Language) (Smith 1992) and ISO 10303 (STEP, STandard for the Exchange of Product data) (Fowler 1995). Both of these are general purpose exchange standards and have defined their generic file format and data (or document) modelling language. The history of SGML dates back to 1969 and SGML was adopted as an ISO standard as early as 1986. SGML is document-oriented and has been successfully applied in document exchange. STEP is targeted at the exchange of data describing products between CAD (Computer Aided Drafting/Design), CAM (Computer Aided Manufacturing) systems and so on. It also aims to be a means of long-term retention of such data (ISO 10303-1:1994). The EXPRESS (ISO 10303-

11:1994) family of information modelling languages is part of the STEP series standards and has found many applications outside STEP. For instance, the ongoing ISO 15046 standards on geographic information use EXPRESS as the lexical schema language (and use Unified Modelling Language as the graphic schema language). However, both SGML and STEP have not been applied as widely as they could have been. Their complexity and the high cost in implementation may have impeded their wide application (Goldfarb and Prescod 2002).

The relatively new XML is a work of W3C, the World Wide Web Consortium. Its aim is to be a data exchange format for the Web. XML is based on and compatible with SGML, but it has been intentionally simplified and made it easy to use. XML 1.0 became a W3C recommendation on February 10, 1998. The most exciting thing about XML is that it has gained possibly the widest ever vendor support and user acceptance and is widely believed to be able to become a universal language for data exchange (Harold 1999, Harold and Means 2002).

5.1.1 The XML Syntax

XML belongs to the markup language category. Markup is a method of conveying meta-data by attaching tags to data items. The tags are often labelled in a lexically meaningful way so as to help data users and processing programs interpret data. Markup has been widely used in data exchange formats to make data self-describing (Goldfarb and Prescod 2002). The various markup methods used are not consistent and a parser for one markup language does not work for another. Figure 5.1 shows different ways of encoding information about a *building* feature. First, a delimited-text encoding is shown in (a), in which no tags are used and the data items are not self-describing. Two non-XML methods of using tags are given in (b) and (c) of

Figure 5.1 to show how diverse the systems of markup can be. Figure 5.1(d) and (e) are XML-style encoding fragments, of which (d) uses non-descriptive tag names while (e) uses descriptive tags.

```
Li Kak Shing Tower, 19, 114.1030, 22.1820, 68m
```

(a) Textual encoding

```
name: Li Kak Shing Tower
stories: 19
longitude: 114.1030
latitude: 22.1820
height: 68
```

(b) Tagged textual encoding

```
Building {
name (Li Kak Shing Tower)
stories (19)
longitude (114.10'30")
latitude (22.18'20")
height (68) }
```

(c) Another way of tagging

```
<tag1>
      <tag2>Li  Kak Shing Tower</tag2>
      <tag3>19</tag3>
      <tag4>114.1030</tag4>
      <tag5>22.1820</tag5>
      <tag6>68m</tag6>
</tag1>
```

(d) The XML tagging method

```
<Building>
      <name>Li  Kak Shing Tower</name>
      <stories>19</stories>
      <longitude>114.1030</longitude>
      <latitude>22.1820</latitude>
      <height>68m</height>
</Building>
```

(e) The eXtensible of XML (Users can introduce tags as needed)

Figure 5.1 XML and other ways of encoding data

The power of XML results mainly from the standard way of using the tags it specifies, the method of tagging into a tree. That is, tags are used in a paired way. Each pair of tags consists of a starting tag and an ending tag. Moreover, tags can be nested as deep as needed but tags can never overlap and thus they form a tree

structure. The tags provide the basis for tag-based data addressing. Most file formats require physical position-based data addressing, i.e. the physical structure of the file must be fixed and the physical position of a data item must be calculated before the data item can be extracted. Being tagged and organized into a tree, data items in XML files have the potential to be addressed by tag name through absolute path or relative path expressions. A file conforming to the XML 1.0 specification (W3C 1998) is called a well-formed XML document. A well-formed XML document simply consists of one (root) document element, which normally has more elements properly nested within each other to form a tree.

5.1.2 XML Vocabulary and XML Schema

XML provides a simple but powerful syntax for encoding data but it does not provide any semantics. Like its ancestor, i.e. SGML, XML does not define any tags for an application domain. XML applications define their own tag sets as needed; that is why XML is called a meta-language for defining markup languages. The process of creating a concrete XML-based markup language or XML application is, by tradition of SGML, called a process of introducing an XML vocabulary. An XML vocabulary defines a set of related tags for use in a class of documents (Birbeck et al. 2001). What underpins the tags is in fact a conceptual model of the application domain. Each application domain or software tool may define its own vocabulary for tagging the documents to be exchanged. This would usually be called a proprietary vocabulary in the sense that other tools would normally not understand the meaning of those tags. XML data, even using a different vocabulary, share the same syntax and can be processed by the same XML parser and other tools. As can be seen, information sharing would benefit from a shared XML vocabulary. Hundreds of

public XML vocabularies have been defined for various application domains (Mulberry Technologies Inc. 2000). CML (Chemical markup language), MathML (mathematics markup language) and BSML (bioinformatic sequence markup language) are among the examples. GML (Geographic Markup Language) is an XML vocabulary and XML-based geodata format from OGC (OGC 2002). ISO TC211 on geographic information also chooses XML as the file format for encoding geodata. Figure 5.2 shows another XML fragment that encodes the same *Building* information that was shown in Figure 5.1. The difference is that the GML vocabulary is used. The use of a vocabulary is indicated by an *xmlns* node of an element. *xmlns* stands for XML Namespace (Skonnard and Gudgin 2001). XML Namespace provides a compound name syntax that extends the definition of XML 1.0 to ensure that unique names can be generated for shared vocabularies. An XML namespace is often (but not necessarily) identified by a URL. In the following example, the namespace prefix `gml` refers to the namespace identified by the URL of `"http://www.opengis.net/gml"`.

```
<Building>
  <name>Li  Kak Shing Tower</name>
  <stories>19</stories>
  <height>68</height>
  <gml:location xmlns:gml="http://www.opengis.net/gml">
    <gml:Point
      srsName="http://www.opengis.net/gml/srs/epsg.xml#4326">
      <gml:coord>
        <gml:X>114.1030</gml:X>
        <gml:Y>22.1820</gml:Y>
      </gml:coord>
    </gml:Point>
  </gml:location>
</Building>
```

Figure 5.2 An example using GML vocabulary

An XML vocabulary is specified using some (schema) language. DTD (Document Type Definition) is a grammar-based language that XML inherits from SGML and

has been widely used for specifying XML vocabularies. A DTD specifies structure of a class of XML documents as well as providing the tag vocabulary for use in these documents. DTD has been designed for specifying document structures and is not well suited for specifying data structures. Most of all, it does not support data types. In addition, it uses pseudo EBNF (Extended Backus-Naur Form) syntax, which is considered hard to use by many. Hence, efforts have been made to design new schema languages for the so-called data-oriented XML documents. Several XML schema languages have been developed, including W3C XML Schema (W3C 1999), RELAX NG (OASIS Technical Committee 2002) and Schematron (Robertsson 2003), etc. For a comparison of them, please refer to Lee and Chu 2000 and Vlist 2001. Among them, W3C XML Schema (WXS) is the response from W3C to the need for a new schema language and is now a W3C recommendation (W3C 1999). The author uses *XML Schema* to refer to the W3C XML Schema recommendation and uses *XML schema* to refer to any schema specified in any schema language designed for describing XML documents. Many XML vocabularies have been or are being converted into XML Schema language from DTD. The significant features of XML Schema include:

- XML Schema supports specifying data types, both simple and complex ones;
- XML Schema supports object-oriented modelling features; and
- XML Schema uses XML syntax instead of EBNF syntax.

An XML schema strictly specifies the structure of one or more XML documents and can be used to validate data. An XML document that conforms to its associated schema is called a *valid XML document*. Figure 5.3 shows the element and type

definitions using XML Schema language for the GML fragment shown in Figure 5.2. For brevity, what is shown is only a schema fragment, with namespace definitions and schema import statements omitted. Also, the definition of `gml:location` is not shown here, which is part of GML vocabulary.

```
<element name=”ex:Building” type=”ex:BuildingType” />

<complexType name="ex:BuildingType">
  <complexContent>
    <extension base="gml:AbstractFeatureType">
      <sequence>
        <element name="ex:name" type=”xs:string”/>
        <element name="ex:stories" type=”xs:integer”/>
        <element name="ex:height" type=”xs:decimal”/>
        <element ref="gml:location"/>
      </sequence>
    </extension>
  </complexContent>
</complexType>
```

Figure 5.3 A sample fragment of XML Schema

As can be seen, an XML vocabulary is specified in one or a group of related schemas. Moreover, the so-called schema is a new XML term used to refer to structural and semantic specifications of XML documents using any definition language, including DTD and XML Schema as well as others. It is thus worth noting the conceptual difference between an XML vocabulary and the set of XML schemas used to specify it. The term *XML vocabulary* has its origin in SGML technology and has been used by inheritance in the XML community. In SGML, a vocabulary refers to the set of tag names defined in one or more DTDs. A vocabulary effectively defines ALL the tag names that can be used in SGML documents of a specific document type. For instance, each version of the well-known HTML (Hyper-Text Mark-up Language) document type has its fixed vocabulary for use in HTML documents. Adding proprietary tags to HTML documents can cause uninformed Web browsers to crash.

As can be observed, in the sense of SGML, a vocabulary is designed for an application domain, not for a specific document or application. In contrast, a data schema in terms of database specifies the structure of a specific dataset or database for a specific application. If XML is used for data representation, an XML data document will normally be specified by a specific XML schema. Although one XML schema can be used for a class of XML data documents, such practice will make an XML data document a mixture of data and meta-data and the data schemas become hidden. In fact, this reflects an essential difference between document modelling and data modelling as will be analysed in detail later. For now, XML documents processing techniques will be examined.

5.1.3 XML Processing Technology

To be effective, a general-purpose data exchange format needs to be an implemented technology instead of a format specification. Further, it is desirable that the data format provides some means of data abstraction. XML has been well implemented. XML editors, parsers, APIs and high-level processing languages are widely available. What is more, these tools conform to standards, i.e. W3C recommendations. In this subsection, relevant XML techniques will be described and their roles in XML-based data exchange analysed.

DOM and SAX

At a low level, there are two standard APIs, DOM and SAX. DOM stands for Document Object Model. Strictly, DOM is not an API but a language and platform neutral specification of API. DOM defines a tree model of XML documents and specifies methods to manipulate the trees (W3C DOM Working Group 2002). SAX stands for Simple API for XML (McLaughlin 2001). It is a public domain API that uses an event-driven method to sequentially access XML document. As SAX API does not require the whole document be loaded into memory, it gives better performance than DOM API. The gain in performance comes at the price of random accessing. Therefore, DOM API is often used where random accessing or modification is required while SAX API is used where sequential accessing is sufficient.

The availability of standard APIs for processing XML documents provides a basic but significant benefit of migrating geodata from proprietary formats to XML format. After converting data into XML format, we do not have to rely on diverse, proprietary and often poorly documented and maintained APIs for accessing geodata in a potentially large number of formats. A common syntax is also the basis for providing commonality at upper levels.

XSLT/XPath

In addition to the low-level DOM API, W3C has defined XML Path language (XPath 1.0:1999), i.e. XPath, to enable parts of XML documents to be addressed and selected. XPath is based on a tree model of XML documents and uses path expression to refer to nodes in the tree. For instance, the XPath expression "Building/gml:Location" may be used to refer to *gml:location* elements in the above sample XML fragment. Currently, there are slight differences between the XPath tree

model and DOM tree model. W3C is working on harmonizing the different tree models of XML documents. XPath is not to be used as a standalone language but is used in other XML data manipulation languages, such as XSLT (XSLT 1.0:1999, XSLT 2.0:2005) and XQuery (XQuery 1.0:2005), as a node addressing and selecting mechanism for the tree representation of XML documents. This study will make heavy use of XSLT/XPath, which is the high-level language that greatly facilitates the transformation of XML documents.

XSLT, an abbreviation for XML Stylesheet Language: Transformation, is a language designed primarily for transforming one XML document into another. XSLT can be considered as the SQL for XML data (Kay 2001). One of the two main scenarios where transformation is needed is to separate data content from data presentation. In a sense, XML is introduced to overcome the limitation of HTML (but XML and HTML are in fact not comparable at all—basically, if XML is a class, then HTML is an instance). In HTML, data and the way of rendering data on devices, such as a Web browser, are mixed or bundled together. That bundling brings difficulties both in further processing the data contained in HTML and in re-rendering data on other devices. The separation of data from its presentation is desired. When rendering instructions are stripped from data, there needs to be a way of turning data into a form ready for presentation, such as HTML. XSLT is used partly for that purpose. It can be used to transform a XML data document into HTML format or other text-based formats. The other main scenario where transformation is needed is data exchange among applications—that is the scenario of concern in this study. The various differences in data organization among different applications have been shown. The tree data model and the high-level XSLT language have in effect provided data abstraction for XML data documents. Hence, XSLT can help reconcile

those differences much more easily than imperative programming. Using XSLT for geodata translation will be discussed in a following section.

Other data-oriented XML techniques

As mentioned above, XML emerges as a common syntax for data exchange. With its conceptual simplicity, XML has gained wide implementation support and application, which again led to the exploration of other potentials of XML. One potential relevant to our topic is the use of XML as a generic data persistence mechanism. XML serialization of objects in the sense of object-oriented programming has been implemented in both Java (McLaughlin 2001) and .Net platforms (Bornstein 2003). XML database technology is emerging. Vendors have been active in providing XML database software (Jagadish et al 2002, Chaudhri et al 2003, Wong 2003, Oracle 2005b). Further, W3C is working on defining a standard query language for XML database, XQuery (XQuery 1.0:2005).

In summary, XML-based technology is becoming a full range solution to data modelling, storage, exchange and management.

5.2 XML and the Proposed Framework

How is XML related to the envisioned framework? As can be seen from the previous section, current XML technology provides a very flexible syntax to encode almost any data, languages for defining XML schemas, and low and high level means to manipulate XML data. Moreover, all these are open standard techniques widely supported on various platforms and by various programming languages and software tools. Therefore, current XML technology has provided the basis for the lower two layers of the proposed framework, namely the data instance and data schema layers.

In particular, XML syntax can be used to encode geodata; XML Schema can be used to specify geodata; and the processing techniques can be used to transform geodata. With respect to the proposed framework, what are still lacking include means for specifying schema mappings, meta-schemas and meta-schema mapping, and techniques to link mappings at different layers together. The rest of this chapter will explore the use of XML technology to implement the lower two layers of the framework in detail. Additional techniques to be developed will be addressed in the next two chapters.

5.3 2-Layer and 3-Layer Models of XML-based Data Formats

Despite its document root, the most common application of XML today deals with data storage within software applications and systems, and data exchange among them (Harold and Means 2002). Both relational and object data can be encoded using XML syntax. For relational data, database vendors have provided both GUI and API ways of converting data between the internal data representation of a database and XML formats. For object persistence of object-oriented programming, XML serialization of objects has been widely adopted.

Due to the document-oriented root of XML, mature methodology for developing XML-based document formats is available. In the methodology, a 2-layer model of document formats is adopted. The two layers are document instance, and document vocabulary and schema. However, the techniques used to build data-centric XML applications vary greatly, depending on the required functionality (Harold and Means 2002). According to the number of layers used in data representation, current methodologies of building data-centric XML applications can be categorized into two groups: those adopting a 2-layer model of XML-based data format and those

adopting a 3-layer model. In the 2-layer model, application data are encoded in XML syntax and at the same time an XML schema is used to specify the data structure and content. In such cases, one single schema applies to all XML data documents of the software system or application. Thus, the schema in fact corresponds to an XML vocabulary in the sense of document-oriented XML applications. This method is appropriate if the stored data are internal data and not intended to be exchanged with other applications. Some XML-based object serialization methods on Java platforms belong to this category. In the 3-layer model, a specific schema is used to specify each specific XML data document. All possible XML schemas of a software system in fact characterize the logical data model of the software system. Some database systems use this means for exporting data to and importing data from XML. Microsoft Access 2002/2003 is such an example. If a meta-schema is formulated using some meta-meta-formalism to specify all possible XML schemas of a software system, then all the involved layers of data representation constitute a 4-layer model of data representation as proposed in Chapter 4.

To compare the 2-layer and 3-layer models for developing XML-based data formats, a sample case of encoding tabular or relational data using these methods is presented in Appendix A to show their differences. The example dataset is a very simple relational database referred to as the `Library` case here. The `Library` database consists of three tables, `book`, `author` and `book_author`, of which `book` contains basic information on books, `author` contains basic information on authors, and `book_and_author` relates books and authors. Each table contains only two or three records to be concise and thus emphasize the conceptual differences between different methods. Part A of Appendix A shows one possible 2-layer method of encoding the `library` database using XML together with its schema. As can be seen,

the schema in fact is not for the `library` database because it is too generic to be specific and cannot validate values of specific data items. For instance, the schema does not and cannot specify that the `id` element of the `author` element should be of data type integer. This kind of dataset level meta-information is interweaved with instance data. The associated software of this format needs to take charge of data validation by itself. In addition, the primary key and foreign key constraints also have to be validated by the software itself. Such a schema is not a true data schema for the dataset but a sort of "document schema" that specifies the structure of "relational documents" encoded this way. Part B of Appendix B shows one possible 3-layer method of encoding the `library` database using XML together with its schema. As can be seen, the schema is very specific for the dataset. The data type of each element is precisely specified. Primary key and foreign key constraints are expressed using the built-in key and key reference mechanisms of XML Schema. The problem with the 3-layer method is that each database/dataset will have a different schema and there is no mechanism in this method to specify what schemas (instead of instance documents) are allowed by the software system. Hence, it cannot easily address the problem of format-to-format data exchange. That is why the 4-layer model of data representation as proposed in Chapter 4 is needed.

Therefore, it can be concluded that the 2-layer model of developing XML applications has been designed for document exchange and hardly suits the purpose of data exchange. If XML-based data formats are developed according to the 2-layer model, the characters of a dataset will be interweaved into instance data and not explicitly stated. The 3-layer method can facilitate data exchange on a dataset-by-dataset basis as each dataset is fully described by its associated schema. However, without explicit meta-schemas, data exchange on the format-by-format basis cannot

be dealt with easily. In the rest of this chapter, the 4-layer method of data representation is assumed but only issues involved in the lower two layers of the 4-layer model will be discussed. The problem of specifying meta-schemas of XML-based data formats will be discussed in Chapter 7.

5.4 Developing XML-based Geodata formats

Converting geodata in its native (non-XML) encoding format to XML format requires the definition of an XML-based data format as a counterpart to the proprietary format and the development of a decoder and encoder between them.

5.4.1 Encoding Geodata using XML Syntax

Geodata formats using non-XML encoding need to be converted into XML format. It is worth noting that the approach of using a common encoding format is different from the idea of using standard geodata transfer formats. A geodata transfer format basically consists of specifications of the data encoding method, the spatial data schema and the feature model and its associated application modelling rules. When XML is used as the common encoding format, the spatial schema and the feature model largely remain untouched. As pointed out in ISO-15046 part-18: encoding, the use of one encoding method does not exclude other encoding methods. The benefit of using XML encoding is to obtain the properties of XML format and leverage XML technology. It is reasonable to believe that more and more GIS vendors will shift to make their data exchange format XML-based. Moreover, vendor-neutral spatial data exchange standards will also add XML encoding or change to use XML encoding. Indeed, it has been reported that the US Census has initiated a project to test XML as the encoding format for their TIGER (Topologically Integrated

Geographic Encoding and Referencing) data (GAI 2004). Furthermore, many new software tools are using XML syntax for data persistence. Section 5.4.4 gives examples of non-XML geodata formats and their XML counterparts.

5.4.2 Using WXS to Represent Spatial Schema and Application Schemas

When converting data to XML format, XML Schema can be used to specify a data schema very close to the original data schema. XML Schema will be used in this study because it is the W3C "official" schema language designed to substitute DTD and has by design taken into account the need of data exchange (Malhotra and Maloney 1999). XML Schema is basically a grammar-based schema language (Fallside 2001). As such, it allows the precise specification of very complex documents and data structures. In the last sub-section, it has been shown that XML Schema provides support for precisely specifying relational data. In addition, XML Schema also aims to describe documents in a manner as close as possible to object-oriented design (Vlist 2001a).

When using XML and XML Schema for encoding and specifying object data, elements can be used to represent objects; both elements and attributes can be used to represent object properties of literal types; object reference(s) can be represented using IDREF and IDREFS attributes; complex types can correspond to object classes; complex type extension/restrictions together with element substitution mechanisms match the inheritance between object classes; object aggregation can be implemented as element nesting. Polymorphism is not relevant here as data, not processing, is being discussed. Hence, it can be said that XML Schema supports the static modelling constructs of the object-oriented model. Therefore, spatial schemas and application schemas of geo-datasets can be translated into XML Schema without

significant semantic loss. In this sense, XML Schema is semantically rich and is suitable to serve as a common schema language for geodata exchange. As an example, the GML vocabulary has been specified using XML Schema. In fact, XML schema is so complex and flexible that various data models can be accommodated in various ways. A detailed discussion of using XML Schema is not appropriate here. Please refer to the W3C XML Schema Recommendation for reference (W3C 2002) or other related books, such as (Vlist 2001a).

5.4.3 Explicating Feature Models as Meta-Schemas

For geodata formats, a logical data model is by convention called a feature model, as a geographic feature (or feature in short) is the central concept in feature- or object-based geospatial modelling. Different feature models of different geodata formats have been compared in Chapter 2. Currently, there is no specially designed meta-meta-formalism in XML technology that can be used to specify feature models/logical data models of geodata/data formats. A method for specifying feature models or logical data models as meta-schemas will be presented in Chapter 7.

5.4.4 The XSHP and XMIF formats

XML counterparts of SHAPE and MIF formats have been developed in this study using the methods discussed above. They are referred to as XMIF and XSHP formats respectively in this writing. Each format definition consists of spatial data type definitions in one XML schema and definitions of abstract feature and feature classes, possibly in a separate XML schema. In addition to these data schema level schemas, the feature model of each format has been specified as a meta-schema, which is also an XML schema. Spatial schema and abstract feature definitions of XSHP and XMIF

are given in Appendix B and C, respectively. Meta-schemas of XSHP and XMIF are given in Appendix D and E, respectively and will be explained in detail in Chapter 7. Here, the focus will be on spatial schemas, abstract feature definitions and application schemas.

XSHP format will be used here for explanation. The definition of XMIF is similar. The namespace "http://www.xggt.org/xshp" is introduced to represent the XSHP vocabulary, of which `xshp` will be used uniformly in this writing as the namespace prefix. Commonly used geometric data types of SHAPE format are specified in schema `xshpTypes3.xsd`. Other geometric types can be defined in the same way. In addition to geometric data types, two extra elements are defined, which are `xshp:XshpFile` and `xshp:_Feature`. Element `xshp:XshpFile` is defined to serve as the root element of an XSHP file. It mainly contains a number of `xshp:_Feature`, which is an abstract element to be substituted by a concrete feature element. An application schema of a specific XSHP file is required to import the `xshpTypes3.xsd` schema. Moreover, the application schema is required to define an element representing a concrete feature class that substitutes the abstract `xshp:_Feature` element. Such constraints on application schemas are specified in the XSHP meta-schema. To illustrate this, a sample XSHP application schema is given in Figure 5.4. The schema imports the `xshpTypes3.xsd` schema to make use of the XSHP vocabulary. It defines a `Lake` feature class having non-geometric fields of `name`, `area`, `surface_elevation` and `depth` as well as a geometric field `Shape` of type `xshp:PolygonPropertyType`. Note that the `lake` feature element and its field elements do not belong to the XSHP vocabulary.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:xshp="http://www.xggt.org/xshp"
xmlns:ns1="http://www.xggt.org/example"
targetNamespace="http://www.xggt.org/example"
elementFormDefault="qualified" attributeFormDefault="unqualified">
  <xs:import namespace=http://www.xggt.org/xshp
  schemaLocation="xshpTypes3.xsd"/>
   <xs:element name="Lake" substitutionGroup="xshp:_Feature">
    <xs:complexType>
     <xs:sequence>
      <xs:sequence>
       <xs:element name="name" type="xs:string"/>
       <xs:element name="area" type="xs:decimal"/>
       <xs:element name="surface_elevation" type="xs:decimal"/>
       <xs:element name="depth" type="xs:decimal"/>
      </xs:sequence>
      <xs:element name="Shape" type="xshp:PolygonPropertyType"/>
     </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

Figure 5.4 A sample XSHP application schema

5.4.5 Developing Decoders/Encoders

Having designed a XML-based encoding method for a geodata format, a decoder must be developed that converts data from the non-XML encoding to the XML encoding. Generally, a decoder is itself a translator in the sense that it converts one representation into another. However, a decoder only changes the encoding format of data without significantly changing its schema. A decoder logically consists of a parser and a converter. The parser reads an input file and constructs an internal representation of the source data. The converter maps each construct of the source encoding format into the corresponding construct of the target XML format. As the converter is a trivial one, the bulk of the work in developing a decoder is the development of the parser. A parser-generating tool may help in developing decoders if the non-XML format is a textual one. Pascoe and Penny (1990) suggest using *yacc* and *lex* to develop decoders for geodata formats. Regarding converting data from a

proprietary file format into XML format, some also suggest the use of parser-generating tools. However, parser-generating tools can be slow to learn and hard to use. For usual data formats, manually developing a parser manually is much easier if the developer is not familiar with the concepts of parser generation. Parser-generating tools remain valuable in developing large data translation systems that involve many file formats.

When developing encoders from an XML format to a proprietary file format, there is no need to develop an XML parser. The previously mentioned DOM and SAX APIs or the equivalents are available on most platforms and for most popular programming languages. In addition to those APIs, XSLT may be used to develop encoders when the target format is a textual one.

Decoders/encoders between SHAPE *and* XSHP and MIF *and* XMIF have been manually developed in this study. XML API on .Net platform has been used for generating XML documents and parsing XML documents. The decoders/encoders have been used in this study to convert test data in SHAPE or MIF format to XSHP or XMIF format, respectively, and vice versa. In addition to converting instance data, the decoders also produce associated XSHP or XMIF schemas.

5.5 Implementing Mechanical Translator using XML Technology

After source data have been converted into XML syntax, the problem of data translation can be viewed as transforming a source XML document to a target XML document. The resulting target XML document will be further processed by encoders to convert it into some target syntax. Generally, custom transformation can be performed on a dataset-by-dataset basis. The process will not be too cumbersome

with the aid of XSLT as will be discussed in the next section. Mechanical translators that transform data from a source XML-based data format to a target XML-based data format are still needed. As the technique of developing mechanical translators has been described in Chapter 3, only how XML technology facilitates the writing of conventional mechanical translators will be shown here. Methods for automatically generating data translators from high-level transformation specifications on both a dataset-by-dataset basis and a format-by-format basis will be addressed in the next two chapters.

If the approach of programming with data accessing API discussed in Chapter 3 is adopted to transform XML geodata documents, DOM or SAX API will be used to parse and process documents with an imperative language, such as C++ and Java. It should be noted there has been significant progress; the widely available standard API such as DOM and SAX can be used due to the adoption of XML syntax. The troubles of dealing with a range of proprietary APIs have been removed. Defining a standard programming interface is a widely adopted approach to facilitate the assembly of software component parts. The STEP series of standards also includes a standard API specification for accessing its file format and has defined mappings to a range of programming languages (Fowler 1995). The task of data schema translation now takes the form of translating one WXS schema to another. That task is facilitated due to the fact that all data schemas are now specified using the same schema definition language, i.e. WXS, and there are community standard APIs for manipulating WXS schemas for various platforms including .NET and Java, etc.

The above discussion shows the benefits of introducing XML as a persistent intermediate syntax. Firstly, the coupling of decoder, translator and encoder is

obviated and thus the development of a translator no longer makes use of, and depends on, a specific decoder or encoder. Secondly, the data definition language and data manipulation techniques are well defined and well supported. As can be seen, when using XML technology as a base, the development of mechanical translators is much facilitated. Moreover, XML technology also makes semantic translation easier, as discussed below.

5.6 Semantic Reconciliation using XSLT

5.6.1 Restructuring and Semantic Reconciliation

As analysed in Chapter 2, the feature models of various geodata formats differ in modelling constructs provided and their application modelling rules. After converting geodata into XML format and using XML Schema to describe their structure, the differences in feature models and the preference of using specific constructs in application schemas exist in the form known as schematic difference, i.e. difference in the structural organization of data. As a language, XSLT is best at manipulating the structure of data (Kay 2001, Mangano 2002).

In terms of implementation, semantic translation means basically to develop for each specific dataset a specific translation program, which translates data in the desired way. As such, implementing semantic translation by procedural programming is cumbersome and error-prone as the majority of the work is the repetition of routine steps. Given the similarity of data transformation cases, it should therefore be possible to describe what to do using a high-level declarative language. As mentioned previously, the XSLT language is one designed to facilitate transforming

XML documents. XSLT can be used for extracting data selectively, reordering data, turning attributes into elements or vice versa, and many other similar tasks.

Compared with transforming XML documents using XML API-s, transforming XML documents with XSLT is an approach at a higher level. However, it still requires detailed programming as will be seen in the next chapter, where a high-level schema mapping method will be developed to allow geodata transformation to be specified declaratively. Its implementation will be based on XSLT.

5.6.2 Integration with Imperative Programming

In the process of geodata translation, it is often the case that some sophisticated computation is required to convert one geometric data type to another geometric data type or to select features based on their derived geometric properties. For example, the conversion of an elliptic arc to a line string is calculation intensive. Another example might be the conversion of only those polygon features that are of spatial significance, with the area chosen as the significance measure, which needs to be calculated from coordinates. As has been mentioned, XSLT is best at manipulating the structure of data rather than data content. That is similar to SQL. Compared with languages like Java or JavaScript (precisely, ECMAScript), it is rather laborious to do complex calculations with XSLT. Here, the concern is not how to do geometric conversion or computation. Rather, it is how to introduce convenient computation functionality into XSLT transformations. It is desired that extension functions written in other languages can be called from within XSLT stylesheets.

The extension mechanism of XSLT provides a basis for meeting the computational requirements in geodata translation. Most XSLT processors provide facilities for

calling extension functions. These mechanisms are, however, implementation-dependant. XSLT 2.0 (XSLT 2.0:2005), which is now in the status of W3C candidate recommendation, provides mechanisms that allow an XSLT stylesheet to determine whether an XSLT implementation makes particular extensions available and to specify what should happen if those extensions are not available. Making use of these mechanisms, it is possible for an XSLT stylesheet to use extension functions and still retain portability (Mangano 2002, XSLT 2.0:2005). In the rest of this study, it is assumed such extension mechanisms are available and can be used to implement portable geodata processing required in geodata translation process.

5.7 Summary

XML technology is becoming a widely accepted general mechanism for data exchange. This chapter explored the application of XML technology to geodata exchange according to the framework proposed in the previous chapter. After a brief introduction to XML and the associated techniques, an overview of using XML for geodata translation was presented. The problem of developing XML-based data formats has been addressed. The 2-layer model of developing XML-based document formats has been shown not to be suitable for developing XML-based data formats. The 3-layer model of developing XML-based data formats has been adopted to develop XSHP and XMIF formats. The 3-layer model will be extended to the 4-layer model of data representation in a later chapter. Methods and benefits of using XML techniques to develop mechanical translators are breifly described. Finally, XSLT together with its extension mechanism is shown to be a good basis for facilitating semantic reconciliation.

From close investigation, it can be concluded that XML technology provides a good technology basis for implementing the framework for geodata exchange proposed in the previous chapter. First of all, with excellent implementation support, XML syntax can serve well as the common syntax for encoding geodata, while WXS serves as the common data definition language for specifying data schemas. Indeed, WXS can also be used to specify meta-schemas as will be seen in Chapter 7. Moreover, the high level XSLT language together with its integration of imperative programming languages offers a flexible means to implement semantic translation. This will become more clear later in the next chapter.

With respect to the proposed framework, what are still lacking include the means for specifying schema mappings, meta-schemas and meta-schema mapping, and techniques to link mappings at different layers together. These are to be discussed in the following two chapters.

As can be seen, in XML-based implementation of the proposed approach, an XML-based counterpart needs to be defined for a non-XML-based data format and a pair of encoder and decoder between them need to be implemented. Data encoded in non-XML-based format need first be converted into the corresponding XML-based encoding. This introduces performance loss In addition, the intermediate XML data can be times larger in file size than the source or resulting dataset as XML is textual and verbose. That means large external storage is needed in the process of data transformation. The trade-off is made for simplicity of translator generation as will be seen in the following chapters.

Chapter 6

Generating Data Instance Translators from Schema Mappings

In the previous chapter, it was shown that after converting data into XML format, a number of standard methods are available for manipulating that data. Among them, XSLT is a high-level language for programming data transformation. However, the use of XSLT for data transformation still requires detailed programming. This is not a trivial operation if it has to be carried out for each data set. Thus, since the introduction of XSLT, many have asked whether it is possible to automate the creation of XSLT stylesheets (DuCharme 2005). It is desirable that, given a source schema and a desired target schema, a program for data transformation can be generated from some simple assertions of correspondence between objects in the source schema and those in the target schema, which this author calls *schema mapping* and is the topic of this chapter.

In the rest of this chapter, an example of data tree transformation using XSLT/XPath is first introduced in Section 6.1. It serves to explain the relationship between tree transformation and schema mapping and the idea of introducing schema mapping on the basis of XSLT/XPath. Section 6.2 gives the schema mapping specification for the example and then generalizes from the exemplary schema mapping a method of schema mapping specification by annotating target schemas. Section 6.3 presents algorithms for translating schema mappings to XSLT transformations. Section 6.4 presents examples of resolving schematic differences in geodata translation using the proposed schema mapping method. Section 6.5 compares the proposed schema

mapping method with related work. Finally, Section 6.6 summarizes this chapter and presents some discussion.

6.1 An Inductive Example of Tree Transformation using XSLT/XPath

This section investigates how data transformation is performed using XSLT/XPath and shows its relationship to the idea of schema mapping.

The two XML documents of concern are shown in Figure 6.1 and 6.2. As can be seen, Figure 6.2 is an XML document where its content is basically the same as that of Figure 6.1. The difference lies in the structure. While document `flat.xml` presents `item`-s in a flat structure and has the child-parent relationship among `item`-s indicated by the `parent` *attribute*, document `tree.xml` presents `item`-s directly in a tree structure, having child `item`-s placed under its parent `item`. Suppose `flat.xml` is the source data set to be converted to a target data set in the form of `tree.xml`.

```
<?xml version="1.0" encoding="UTF-8"?>
<root xmlns="mapping:examples-itemFlat">
      <item id="1" parent="0"/>
      <item id="2" parent="1"/>
      <item id="3" parent="2"/>
      <item id="4" parent="1"/>
      <item id="5" parent="2"/>
      <item id="6" parent="5"/>
</root>
```

Figure 6.1 `flat.xml`

```
<?xml version="1.0" encoding="UTF-8"?>
<root xmlns="mapping:examples-itemTree">
      <item id="1">
            <item id="2">
                  <item id="3"/>
                  <item id="5">
                        <item id="6"/>
                  </item>
            </item>
            <item id="4"/>
      </item>
</root>
```

Figure 6.2 `tree.xml`

The method of XSLT is that of tree transformation, where one tree structure is transformed into another. Different tree data models can be associated with XML documents. In the W3C series recommendations, XPath specification (XPath 1.0:1999, XPath 2.0:2005) defines a tree data model for XML documents. By parsing an XML document, a labelled data tree can be obtained. The corresponding data trees of documents `flat.xml` and `tree.xml` are shown in Figure 6.3 and 6.4, respectively, of which the presentation scheme is borrowed from Kay (Kay 2001). With the associated tree structures, the problem of XML data conversion can be treated as transforming one data tree into another. This idea is illustrated in Figure 6.5.

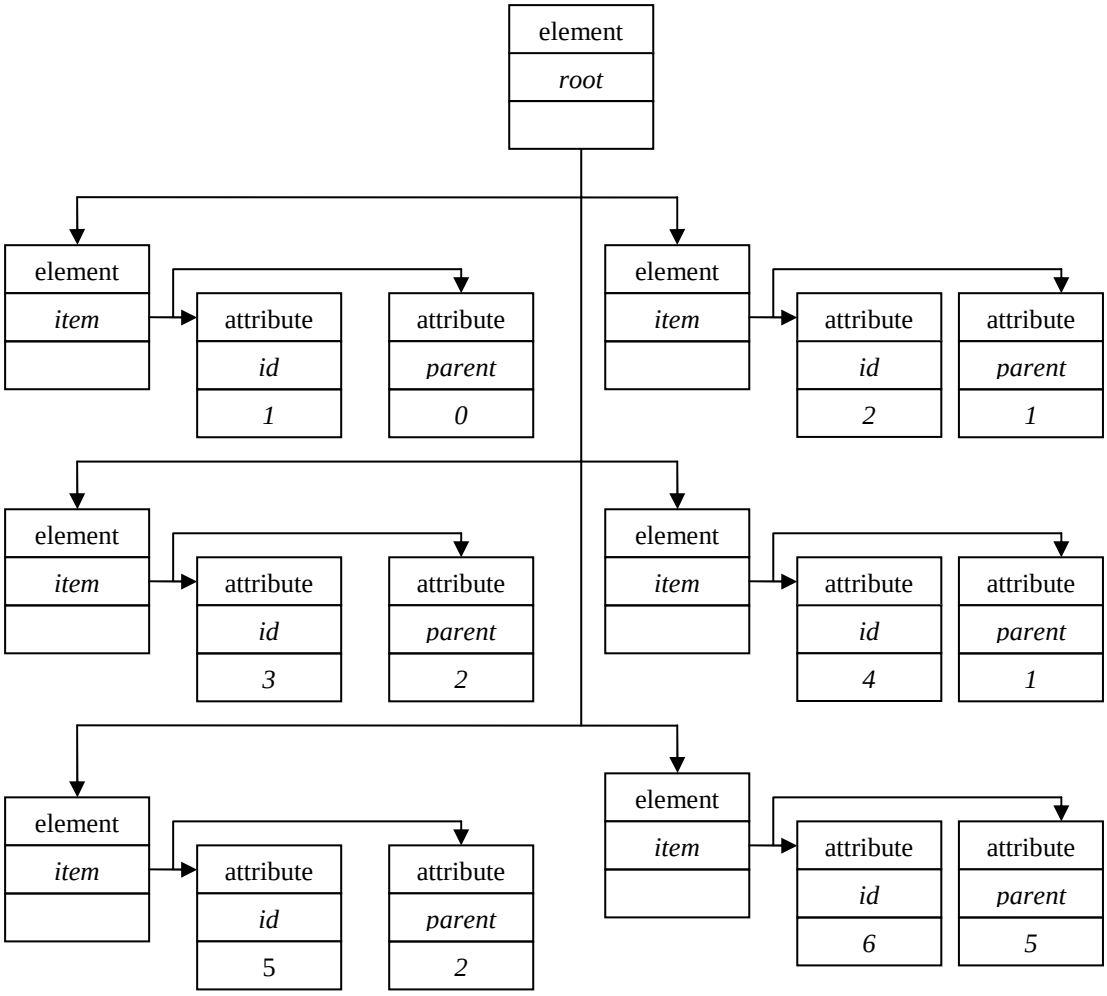


Figure 6.3 Tree structure of `flat.xml`

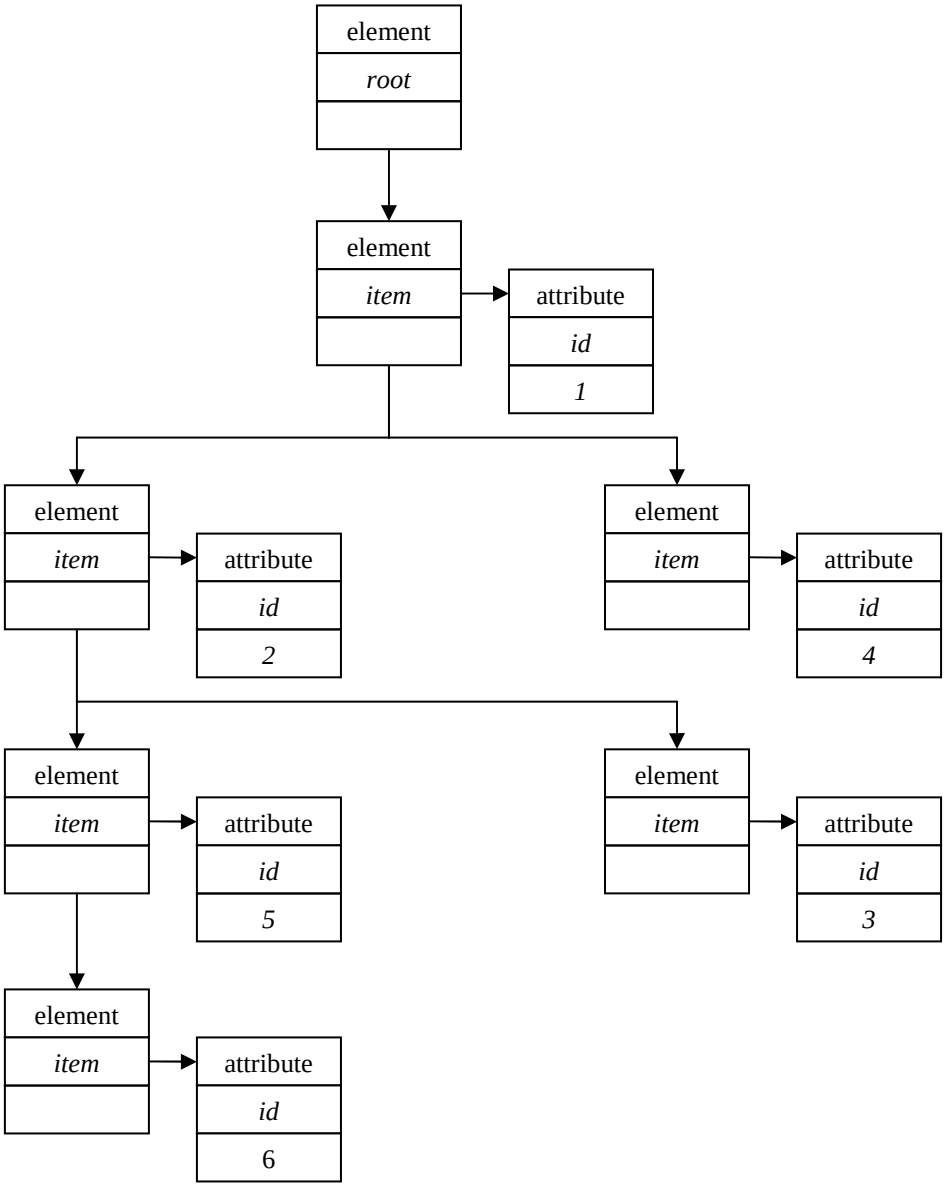


Figure 6.4 Tree structure of `tree.xml`

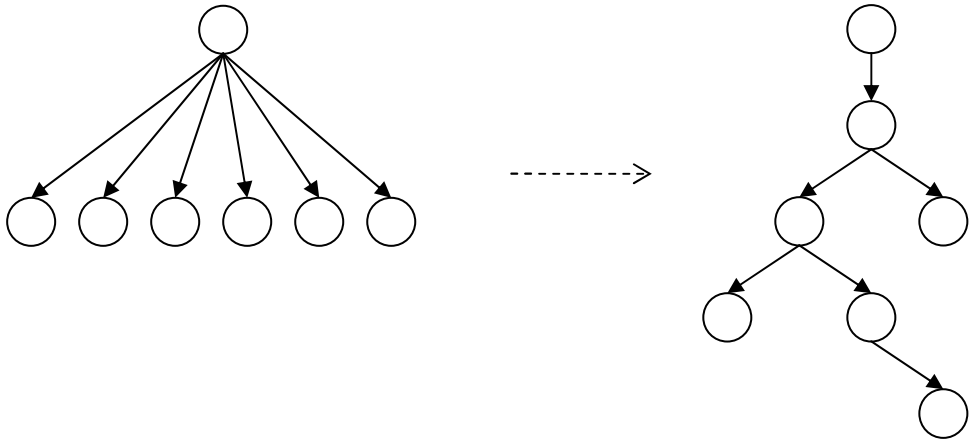


Figure. 6.5 Tree transformation – an illustration

To perform tree transformation, several basic issues need to be addressed. First, it is necessary to (provide a means to specify and) specify which source nodes are corresponding to which target nodes. Secondly, the method of assembling the target tree with (translated) nodes from the source tree must be specified. Thirdly, it is necessary to specify how to find the source node corresponding to a target node.

The XSLT code for transforming the source `flat.xml` to the target `tree.xml` is given in Figure 6.6 to illustrate how XSLT works. As can be seen, an XSLT stylesheet consists of a number of *templates*, which are counterparts of *procedures* in imperative programming languages. In the process of execution, a stylesheet can be considered to walk through the source tree searching for source nodes to be processed, which it then processes and generates the target tree. Each template processes some nodes from the source tree and transforms them into corresponding target nodes. There are two categories of *templates*, called named and not-named. A not-named *template* does not have a name and thus cannot be referred to and called explicitly. Instead of a name, it has a *match pattern* that specifies the conditions under which it will be fired. The XSLT engine will fire the *template* automatically if the conditions are satisfied. A named *template* without a *match pattern* attribute will

not fire itself automatically but be fired on call. So, *templates* are chained one after another implicitly by *pattern* matching or explicitly by *template* calls. This means that the execution flow of a XSLT stylesheet can be determined partially at the time of coding, but also depends on the input data at run time.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
xmlns:tree="mapping:examples-itemTree"
xmlns:flat="mapping:examples-itemFlat">

  <xsl:output method="xml" version="1.0" encoding="UTF-8"
indent="yes"/>

  <xsl:template match="/">
    <xsl:apply-templates/>
  </xsl:template>

  <xsl:template match="flat:root">
    <tree:root>
      <xsl:apply-templates select="flat:item (@parent='0')"/>
    </tree:root>
  </xsl:template>

  <xsl:template match="flat:item">
    <tree:item id="{@id}">
      <xsl:apply-templates select="../flat:item
      (@parent=current()/@id)"/>
    </tree:item>
  </xsl:template>

</xsl:stylesheet>
```

Figure 6.6 A hand-coded XSLT stylesheet transforming `flat.xml` into `tree.xml`

In the code shown in Figure 6.6, the chaining of templates is half implicit in the sense that a *template*, after processing a matched node, explicitly specifies what nodes should be processed next while not specifying which *template* is to be used to process them. An XSLT stylesheet equivalent to the one shown in Figure 6.6 is shown in Figure 6.7, which uses explicit *template* calls instead. In Figure 6.6, the first template states that if a `flat:root` is found, a `tree:root` should be generated and then the root `flat:item found`, i.e. a `flat:item` whose `parent` attribute equals 0, and processed. The second template states that if a `flat:item` is found, a

`tree:item` is generated and its child `item`-s found, i.e. `item`-s whose `parent` attribute equals `id` attribute of the current `item`, and processed. The potential benefit of not specifying which *template* to use is that behaviour polymorphism can be implemented. The processor can choose at run time which *template* to fire, depending on the specific properties of each node being processed.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:fn="http://www.w3.org/2005/xpath-functions"
xmlns:xdt="http://www.w3.org/2005/xpath-datatypes"
xmlns:tree="mapping:examples-itemTree"
xmlns:flat="mapping:examples-itemFlat">

  <xsl:output method="xml" version="1.0" encoding="UTF-8"
indent="yes"/>

  <xsl:template match="/">
    <xsl:for-each select="flat:root">
      <xsl:call-template name="tree:root"/>
    </xsl:for-each>
  </xsl:template>

  <xsl:template name="tree:root">
    <xsl:element name="tree:root">
      <xsl:for-each select="flat:item (@parent='0')">
        <xsl:call-template name="tree:item"/>
      </xsl:for-each>
    </xsl:element>
  </xsl:template>

  <xsl:template name="tree:item">
    <xsl:element name="tree:item">
      <xsl:attribute name="id">
        <xsl:value-of select="@id"/>
      </xsl:attribute>
      <xsl:for-each select="../flat:item  (@parent=current()/@id)">
        <xsl:call-template name="tree:item"/>
      </xsl:for-each>
    </xsl:element>
  </xsl:template>

</xsl:stylesheet>
```

Figure 6.7 An XSLT stylesheet equivalent to that of Figure 6.6 using explicit calls

How the above-mentioned issues of tree transformation are addressed in the XSLT/XPath combination will now be discussed. XPath is designed to tackle the

third problem. As shown earlier, it provides a way of addressing a data tree. XSLT provides a way of dealing with the first and the second problems. In a XSLT stylesheet, the correspondences between source nodes and target nodes are specified implicitly by programming. For instance, in the example currently being discussed, the following statements implicitly specify that a `tree:root` corresponds to a `flat:root`:

```
<xsl:template match="flat:root">
  <tree:root>
 <xsl:apply-templates select="flat:item (@parent='0')"/>
  </tree:root>
</xsl:template>
```

The problem of how to assemble the target tree with translated nodes is addressed through nesting *template* calls or applications in target *elements*. In the above lines of code, the template application is placed within the body of the `tree:root` *element*, indicating that the result of that *template* application will be taken as child nodes of the enclosing *element*, i.e. `tree:root`. In a chain of recursive *template* calls or applications, the results of deeper *template* calls or applications are taken as children of the immediate upper level *template*. In the following lines of XSLT code, the results of inner *template* applications will be taken as *children* of *elements* produced in the outer *template*.

```
<xsl:template match="flat:item">
  <tree:item id="{@id}">
    <xsl:apply-templates select="../flat:item
    (@parent=current()/@id)"/>
  </tree:item>
</xsl:template>
```

As can be seen, as well as specifying *nodes* correspondence, the main task in programming XSLT for data transformation is to take care of the chaining of

templates, either implicitly or explicitly, so that the desired tree structure can be generated correctly.

Although XSLT together with XPath is a powerful language for transforming XML data, it does not make use of the data schema to simplify data transformation. It is worth noting that the tree structure of an XML document is specified by its schema. In particular, the target tree structure is specified by the target schema. Therefore, a target schema, once defined, can be used to guide the transformation process, i.e. to control the chaining of *templates*. That means it is possible to derive an XSLT transformation or stylesheet from correspondence or mapping specification between objects in the source schema and target schema, which is called schema mapping here.

6.2 WSX-based Schema Mapping Specification

6.2.1 Schema Mapping of the Illustrative Example

The purpose is to develop a method of specifying schema mapping. Continuing the exploration with the above example, Figures 6.8 (a) and (b) show the schemas of `flat.xml` and `tree.xml`, respectively and their graphical representations are shown in Figures 6.9 (a) and (b), respectively. With the data schemas, it is required to specify object correspondences among them and have a transformation program generated.

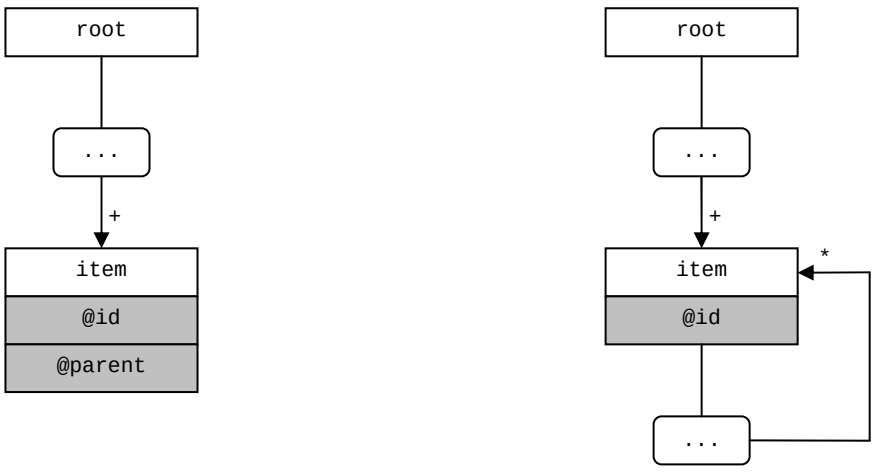
```
<?xml version="1.0" encoding="UTF-
8"?>
<xs:schema
xmlns:xs="http://www.w3.org/2001/XM
LSchema" xmlns="mapping:examples-
itemFlat"
targetNamespace="mapping:examples-
itemFlat"
elementFormDefault="qualified"
attributeFormDefault="unqualified">


  <xs:element name="root">
    <xs:complexType>
      <xs:sequence>
       <xs:element ref="item"
        maxOccurs="unbounded"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>

  <xs:element name="item">
    <xs:complexType>
      <xs:attribute name="id"
       type="xs:integer"/>
      <xs:attribute name="parent"
       type="xs:integer"/>
    </xs:complexType>
  </xs:element>

</xs:schema>
```

(a)

```
<?xml version="1.0" encoding="UTF-
8"?>
<xs:schema
xmlns:xs="http://www.w3.org/2001/XM
LSchema" xmlns="mapping:examples-
itemTree"
targetNamespace="mapping:examples-
itemTree"
elementFormDefault="qualified"
attributeFormDefault="unqualified">

  <xs:element name="root">
    <xs:complexType>
      <xs:sequence>
       <xs:element ref="item"
        maxOccurs="unbounded"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>

  <xs:element name="item">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="item"
         minOccurs="0"
         maxOccurs="unbounded"/>
      </xs:sequence>
      <xs:attribute name="id"
       type="xs:integer"/>
    </xs:complexType>
  </xs:element>

</xs:schema>
```

(b)

Figure 6.8 The `flat.xsd` (a) and `tree.xsd` (b) schemas



(a) (b)

Figure 6.9 Graphic representations of the flat.xsd (a) and tree.xsd (b) schemas

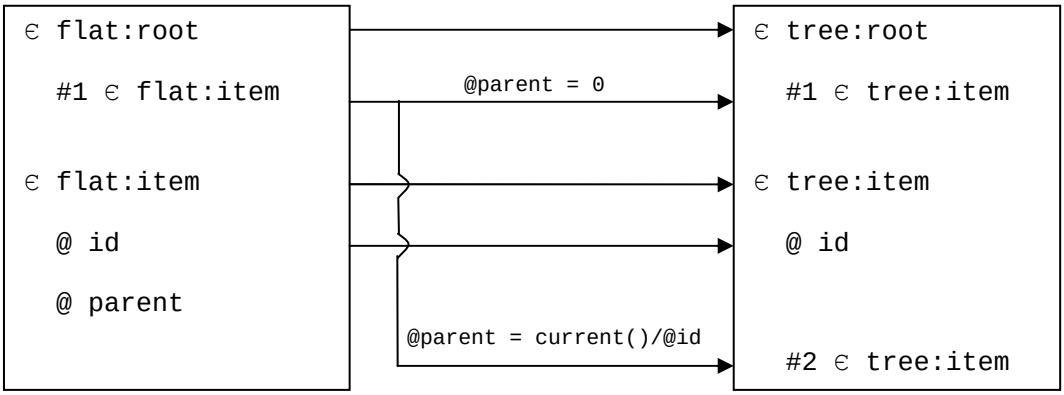


Figure 6.10 Graphical representation of schema mapping
(∈ stands for element; @ stands for attribute; # stands for reference)

Figure 6.10 gives a graphical representation of schema mapping from `flat.xsd` to `tree.xsd`. Intuitively, it says:

a `flat:root` *element* maps to a `tree:root` *element*;

a `flat:item` *element* under `flat:root`, of which `@parent` equals `0`, maps to a `tree:item` under `tree:root`

a `flat:item` *element* generally maps to a `tree:item`

the `id` *attribute* of a `flat:item` *element* maps to the `id` *attribute* of a `tree:item`

a `flat:item`, of which `@parent` equals `@id` of the current `flat:item`, maps to a `tree:item` under the current `tree:item`.

As can be seen from Figure 6.10, the schema mapping strictly follows the target schema. Thus, we can derive a target schema-centred textual representation of schema mapping. A textual schema mapping specification corresponding to that of Figure 6.10 is shown in Figure 6.11, which shows that the textual schema mapping is basically the target schema annotated with asserted correspondences from a source

schema. In particular, the following observations are made. First, an `xs:annotation` *element* is added to `xs:schema` *element* which specifies the source schema of the mapping. Then, it can be noticed that each object (i.e. *element* and *attribute*) of the target schema is annotated with an `xsm:source` *attribute*, which specifies the corresponding object in the source schema.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:xsm="http://www.xggt.org/mapping"
xmlns:tree="mapping:examples-itemTree"
xmlns:flat="mapping:examples-itemFlat"
targetNamespace="mapping:examples-itemTree"
elementFormDefault="qualified"
attributeFormDefault="unqualified">

   <xs:annotation>
      <xs:appinfo>
         <xsm:sourceSchema schemaLocation="flat.xsd"/>
         <xsm:root name="tree:root"/>
      </xs:appinfo>
   </xs:annotation>

   <xs:element name="root" xsm:source="flat:root">
      <xs:complexType>
         <xs:sequence>
          <xs:element ref="tree:item" maxOccurs="unbounded"
          xsm:source="flat:item (@parent=0)"/>
         </xs:sequence>
      </xs:complexType>
   </xs:element>

   <xs:element name="item" xsm:source="flat:item">
      <xs:complexType>
         <xs:sequence>
          <xs:element ref="tree:item" minOccurs="0"
          maxOccurs="unbounded" xsm:source="../flat:item
          (@parent=current()/@id)"/>
         </xs:sequence>
         <xs:attribute name="id" type="xs:integer"
         xsm:source="@id"/>
      </xs:complexType>
   </xs:element>

</xs:schema>
```

Figure 6.11 A sample schema mapping `tree.xsm.xsd` that specifies mapping from `flat.xsd` to `tree.xsd`

6.2.2 Specifying Schema Mapping by Annotating

Before moving on to translating the schema mapping presented above to XSLT, an elaborate description of schema mapping by means of annotating a target schema is needed. W3C XML Schema (WXS) is well known for its complexity as an XML schema language (Kawaguchi 2001, Vlist 2001a). A mapping language that fully supports WXS is considered to deserve separate studies. As the purpose here is to demonstrate the idea of schema mapping, the discussion will only deal with a subset of WXS. The subset is basically the grammar and type specification part of WXS while its object-oriented features are largely left out. In particular, the following WXS constructs with their respective limitations are supported.

- ✓ Global and local *element*
- ✓ *element ref* (i.e. reference)
- ✓ Global and local *complexType*
- ✓ *sequence* of exact one occurrence (which is the default case)
- ✓ *choice* of exact one occurrence (which is the default case)
- ✓ *all* (which is by definition required to be of exact one occurrence)
- ✓ *abstract element* together with *substitutionGroup* of one level depth

Constructs not considered are:

- ✗ `extension/restriction` of *complexType*
- ✗ Arbitrary depth of *substitutionGroup*

- *sequence/choice* of arbitrary cardinality

The restriction of exact one occurrence of *sequence/choice* is due to implementation considerations. As has been mentioned, the target language of schema mapping will be XSLT. Unfortunately, there does not seem to be a way to deal with an arbitrary occurrence of *sequence/choice* using XSLT. XSLT/XPath *pattern* matching and node selection is based on *node name, node content, node document position, node type* and *node data type*. It is extremely inconvenient and inefficient, if possible at all, to identify a specific occurrence of a *sequence* or *choice* particle from a list of them, given that within a *sequence* or *choice* there can be particles of arbitrary occurrence. An example schema of such case is shown below in Figure 6.12.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  elementFormDefault="qualified"
  attributeFormDefault="unqualified">
  <xs:element name="a">
    <xs:complexType>
      <xs:sequence maxOccurs="3">
        <xs:element name="b" maxOccurs="4" type="xs:integer"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

Figure. 6.12 A sample schema allowing multiple occurrences of `sequence`

```
<?xml version="1.0" encoding="UTF-8"?>
<a xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="s.xsd">
  <b>0</b>
  <b>1</b>
  <b>2</b>
  <b>3</b>
  <b>4</b>
  <b>5</b>
</a>
```

Figure. 6.13 A sample instance file conforming to the schema in Figure 6.12

A sample instance file conforming to schema of Figure 6.12 is shown in Figure 6.13. As can be seen, there is no deterministic way to parse the instances according to the schema. That makes the practical use of arbitrary occurrence of *sequence/choice* problematic. As evidence of that, the latest (at the time of writing) release of the well-known MapForce software tool from Altova GmbH cannot deal with this problem. As far as this author is aware, there are no XSLT-based transformation-generating tools that can deal with the problem at hand. From the viewpoint of data modelling, this restriction is not a big loss. First, it is not considered good practice to identify items by *position* instead of *name*. Second, when repetition of *sequence/choice* is needed, there is always a workaround of adding an *element* that encompasses the repetition.

A schema mapping *namespace* and its associated vocabulary are introduced. Its URI is “http://www.xggt.org/mapping”. The *namespace prefix* `xsm` will be used uniformly in this writing to denote this *namespace* and `xs` denotes that of W3C XML Schema.

A schema mapping from a source schema to a target schema is the target schema annotated with mapping information using vocabulary defined in the `xsm` *namespace*. The `xsm` vocabulary consists of the following items.

At the `schema` level, there are annotation *elements* as follows.

Element `xsm:sourceSchema` specifies the source schema of the mapping. It should be included in a top-level `xs:annotation/xs:appinfo`.

Element `xsm:import` specifies an imported schema mapping. It should be included in a top-level `xs:annotation/xs:appinfo`. There can be as many `xsm:import` *elements* as needed.

Element `xsm:root` is used to specify the *root element* of the target XML document. It is needed because there is no means in WXS to specify the *root element* of an XML document, while a transformation has to start from the *root element* and then go recursively downwards.

At `element/attribute/type` level, the following annotation *attributes* and *elements* apply.

Attribute `xsm:source` specifies the source of a target object. It is to be used with local `xs:element`, local `xs:attribute`, `xs:element ref`, `xs:attribute ref`, global `xs:element`, global `xs:complexType` and global `xs:simpleType`. For local target objects, its `xsm:source` is either an XPath *path expression* that selects nodes from the source data tree, if any, or an empty string. For global target objects, its `xsm:source` *attribute* is either the `name` *attribute* of the corresponding source global object, if any, or an empty string. A target object without `xsm:source` *attribute* will be ignored in the process of translating a schema mapping to a data translator.

Attribute `xsm:method` specifies the method of type conversion. It is to be used together with `xsm:source` *attribute*. By default, the target object will be derived from the source object by implicit type conversion. In the case of converting a value of a *simple type* to that of a compatible *simple type*, the default method is to copy the value of the source object to the value of the target object. When the source and target objects are of incompatible data types, an explicit type conversion method needs to be specified. In cases involving *complex types*, the default behaviour of mapping is structural transformation in a recursive way, as will be seen in the next section. When a deviation from the default structural transformation is desired, `xsm:method` *attribute* can be used to specify the particular method of structural

transformation. For the current implementation, a method needs to be a named XSLT template and the `xsm:method` *attribute* gives the `name` of the *template*. Accordingly, there should be an XSLT *import* statement in the top-level `xs:annotation/xs:appinfo` *element* of the schema mapping, indicating where the associated XSLT stylesheet is located.

Element `xsm:variable` specifies a variable to be used within a global mapping by its local mappings. There are cases in which a mapping contained within a complex mapping needs to use as its `xsm:source` *attribute* an XPath expression containing information out of its own *context*, such as `position()` of the parent node. In such cases, the out-of-context information needs to be passed using variables. An `xsm:variable` has a `xsm:source` *attribute*. It should be included in a `xs:annotation/xs:appinfo` *element*.

Element `xsm:param` specifies any parameter required in a global mapping. Its usage is similar to `xsm:variable` except that it is used to pass out-of-context information to a global mapping instead of a local mapping. It should be included in a `xs:annotation/xs:appinfo` *element*. It can have an `xsm:source` *attribute* to serve as a default value for the parameter.

Element `xsm:with-param` specifies parameters to be passed to a global mapping. It is the counterpart of `xsm:param` *element* used with *element references* or *elements* of global *complexType-s*. An `xsm:with-param` has a `xsm:source` *attribute*.

6.3 Translating Schema Mappings to Data Instance Translators

A schema mapping specification needs to be translated into a program that can transform a source data tree into a target data tree. The target code of a schema

mapping can be, in principle, in any programming language. Here XSLT is chosen as the target language for its high-level features, which can save a lot of programming effort. The algorithm for translating schema mapping specifications into XSLT stylesheets is as follows.

6.3.1 The Algorithms

Plain global element

Input*:*

A global *element* with an `xsm:source` *attribute*, which is not *abstract* and not a *substitution*, or:

```
<xs:element name=″eName″ … xsm:source=″*″>
  ...
</xs:element>
```

Processing*:*

Generate a named *template*, the `name` of which is that of the input *element*
    Generate a new *element* of the input kind
    Do element type processing (see below)

Output*:*

a named *template*, a new *element* of the input kind, and the result of element type processing (see below). Or:

```
<xsl:template name="eName">
  <xsl:element name="eName">
    <!-- result of element type processing -->
    <!-- result of successive processing ... -->
  </xsl:element>
</xsl:template>
```

Element type processing

Input*:*

elements of global type. Or,

```
<xs:element … type=″eType″>
  ...
</xs:element>
```

Processing*:*

Case = eType

simpleType:

Generate XSLT code that copies the value of source object.

complexType

Generate a call to the template associated with the *complexType*.

Output*:*

Case = eType

simpleType:

```
<xsl:value-of select="."/>
```

complexType

```
<xsl:call-template name=″eType″/>
```

Global substitute element

Input*:*

a global *element* with an `xsm:source` *attribute*, which is a *substitute* to a *group*. Or:

```
<xs:element name=″eName″ … substitutionGroup=″sGroup″
xsm:source=″sName″>
  ...
</xs:element>
```

Processing*:*

Generate a *template*, the *mode* `name` of which is that of the `substitutionGroup` and its matching *pattern* equals the name of the source object

Generate a new *element* of the input kind

Do element type processing

Output*:*

a *template* with `match` and `mode` *attributes*, a new *element* of the input kind, and the result of element type processing. Or:

```
<xsl:template match="eName" mode=″sName″>
  <xsl:element name="eName">
    <!-- result of element type processing -->
    <!-- result of successive processing ... -->
  </xsl:element>
</xsl:template>
```

Global complexType

Input*:*

a global `complexType` with an `xsm:source` *attribute*. Or:

```
<xs:complexType name=″tName″ … xsm:source=″*″>
   ...
</xs:complexType>
```

Processing*:*

Generate a named *template*, the`name` of which is that of the `complexType`
Process attributes

Output*:*

a named *template* and the result of attribute processing. Or:

```
<xsl:template name="tName">
   <!-- the result of attribute processing (see below) -->
   <!-- result of successive processing ... -->
</xsl:template>
```

Local complexType

Input*:*

a local `complexType`, of which the associated element has an `xsm:source` attribute, or:

```
<xs:element name=”*” xsm:source=”*”>
  <xs:complexType>
    ...
  </xs:complexType>
</xs:element>
```

Processing*:*

Process attributes

Output*:*

the result of attribute processing, if any, or:

```
<!-- xsl:template name="*" -->
   <!-- the result of attribute processing (see below) -->
   <!-- result of successive processing ... -->
<!-- /xsl:template -->
```

Attribute

Input:

an `attribute` *element* with an `xsm:source` *attribute*, or:

```
<xs:attribute name=”aName” … xsm:source=”sPath”>
  ...
</xs:attribute>
```

Processing:

Generate an `attribute`, the `name` of which is that of the input `attribute` *element*
Copy the value of the source object

Output:

```
<xsl:attribute name="aName">
   <xsl:value-of select=”sPath”/>
</xsl:attribute>
```

Element reference

Input:

a *reference* to a global *element* with a `xsm:source` *attribute*, or:

```
<xs:element ref=″eName″ … xsm:source=″sPath″/>
```

And the referred *element* may be one of the following:

```
<xs:element name=″eName″ …xsm:source=″*″/>
<xs:element name=″eName″ abstract=″true″ … xsm:source=″*″/>
<xs:element name=″eName″ substitutionGroup=″sGroup″ …
xsm:source=″*″/>
```

Processing:

case = referred element

abstract:

generate an `apply-template`, of which the `mode` is the `name` of the referred *element* and the *selection* is as the specified source objects. Or:

substitute:

generate an `apply-template`, of which the `mode` is the `name` of the `substitutionGroup` of the referred *element* and the *selection* is as the specified source objects. Or:

plain:

Generate an `for-each` loop, which selects source objects as specified in the `xsm:source` attribute.

Output:

case = referred element

abstract:

```
<xsl:apply-templates select="sPath" mode=″eName″>
```

substitute:

```
<xsl:apply-templates select="sPath" mode=″sGroup″>
```

plain:

```
<xsl:for-each select=″sPath″>
  <xsl:call-template name="eName">
</xsl:for-each>
```

Local element

Input:

An local *element* with an `xsm:source` *attribute*, or:

```
<!-- compositor -->
  <xs:element name=”eName” … xsm:source=”sPath”/>
<!-- compositor -->
```

Processing:

> Generate a `for-each` loop, which *selects* source objects as specified in the `xsm:source` *attribute*.

Output:

```
<xsl:for-each select=”sPath”>
  <xsl:call-template name="eName">
</xsl:for-each>
```

6.3.2 An Illustration of Translating Schema Mappings to XSLT Stylesheets

Given the schema mapping specification method and the algorithms for translating schema mappings to XSLT stylesheets, a schema mapping processor has been implemented using XSLT. The schema mapping processor takes as input a schema mapping and generates as output an XSLT stylesheet. The latter is a data translator that translates or transforms source data conforming to the source schema into data conforming to the target schema. This is illustrated in Figure 6.14.

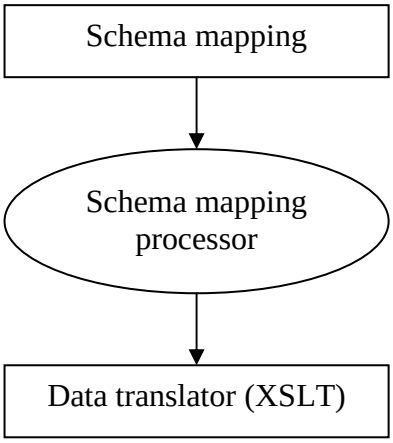


Figure 6.14 Schema mapping processor

To further illustrate the translation process, an example of translating a schema mapping to an XSLT stylesheet is presented here. The example is a continuation of the previously used example of translating `flat.xml` to `tree.xml`. The input schema mapping `tree.xsm.xsd` is shown in Figure 6.11. The derived XSLT stylesheet `tree.xsm.xslt` is shown in Figure 6.15. Figure 6.16 gives the step-by-step illustration.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet xmlns:flat="mapping:examples-itemFlat"
xmlns:fn="http://www.w3.org/2005/xpath-functions"
xmlns:tree="mapping:examples-itemTree"
xmlns:xdt="http://www.w3.org/2005/xpath-datatypes"
xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
xmlns:xsm="http://www.xggt.org/mapping" version="2.0">
 <xsl:output method="xml" version="1.0" encoding="UTF-8"
 indent="yes"/>
 <xsl:template match="/">
  <xsl:for-each select="flat:root">
   <xsl:call-template name="tree:root"/>
  </xsl:for-each>
 </xsl:template>
 <xsl:template name="tree:root">
  <xsl:element name="tree:root">
   <xsl:for-each select="flat:item (@parent=0)">
    <xsl:call-template name="tree:item"/>
   </xsl:for-each>
  </xsl:element>
 </xsl:template>
 <xsl:template name="tree:item">
  <xsl:element name="tree:item">
   <xsl:attribute name="id">
    <xsl:value-of select="@id"/>
   </xsl:attribute>
   <xsl:for-each select="../flat:item (@parent=current()/@id)">
    <xsl:call-template name="tree:item"/>
   </xsl:for-each>
  </xsl:element>
 </xsl:template>
</xsl:stylesheet>
```

Figure 6.15 XSLT stylesheet `tree.xsm.xslt` generated from schema mapping `tree.xsm.xsd`

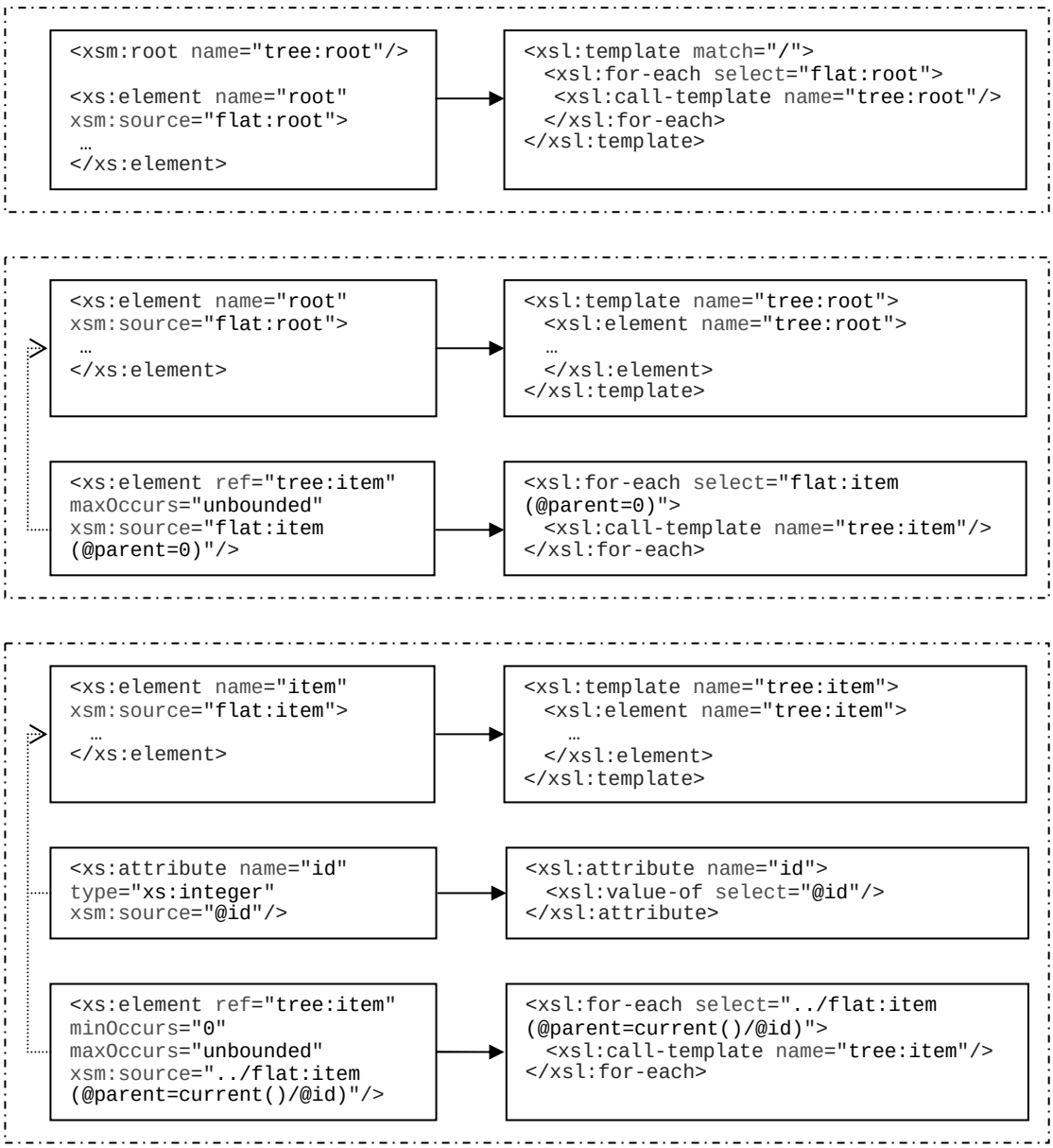


Figure 6.16 Step-by-step illustration of translating schema mapping into XSLT

6.4 Application Examples

It should now be clear that the purpose of introducing schema mapping is to enable high-level specification and customization of geodata translators as a means of facilitating semantic geodata translation. Schema mapping is to be used in two situations. First, a target schema has been defined and the translation needs to produce target data accordingly from source data. Secondly, given a default schema

mapping derived from translating a source schema to a target schema, a human operator can carry out customization so that the transformed data suits the target application better. In the next chapter, the derivation of a default schema mapping will be discussed. Here, two examples are presented to show how schema mapping can facilitate customizing geodata translation.

In the first example, PointM-typed `Building` features of XSHP are to be transformed to XMIF format. The PointM geometric type can be used to model a point feature that has an associated *measure*, which has no exact correspondence in MIF format. Generally, the *measure* (sub-)attribute of the PointM-typed attribute needs to be transformed to some attribute of a MIF feature and the *point* (sub-)attribute to a MIF point. This example shows how that can be done with schema mapping. Figure 6.17 shows the schema (`xshpBuildingPointM.xsd`) of the source XSHP file. The imported schema defining XSHP geometric data types (`xshpTypes3.xsd`) is omitted for brevity. Figure 6.18 shows the schema mapping `xmifBuildingPoint.xsm.xsd` from `xshpBuildingPointM.xsd` to `xmifBuildingPoint.xsd`. The imported schema mapping has been omitted for brevity. Figure 6.19 shows the generated XSLT stylesheet `xmifBuildingPoint.xsm.xslt`. With the schema mapping method, customization of the name, scale factor and data type of `measure` can be easily done; that is not possible using conventional black-box translators.

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- edited with XMLSPY v5 rel. 3 U (http://www.xmlspy.com) by
ABCD (XYZ) -->
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:fr="http://www.xggt.org/xshp/example-buildingPointM"
xmlns:xshp="http://www.xggt.org/xshp"
targetNamespace="http://www.xggt.org/xshp/example-buildingPointM"
elementFormDefault="qualified"
attributeFormDefault="unqualified">
  <xs:import namespace="http://www.xggt.org/xshp"
  schemaLocation="../../xshp/xshpTypes3.xsd"/>

  <xs:element name="Building" substitutionGroup="xshp:_Feature">
    <xs:complexType>
      <xs:sequence>
        <xs:sequence>
          <xs:element name="name" type="xs:string"/>
          <xs:element name="stories" type="xs:integer"/>
          <xs:element name="usage" type="xs:string"/>
        </xs:sequence>
        <xs:element name="Shape" type="xshp:PointMPropertyType"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>

</xs:schema>
```

Figure 6.17 Schema `xshpBuildingPointM.xsd`

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- edited with XMLSPY v5 rel. 3 U (http://www.xmlspy.com) by
ABCD (XYZ) -->
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:to="http://www.xggt.org/xmif/example-buildingPointM"
xmlns:fr="http://www.xggt.org/xshp/example-buildingPointM"
xmlns:xshp="http://www.xggt.org/xshp"
xmlns:xsm="http://www.xggt.org/mapping"
xmlns:xmif="http://www.xggt.org/xmif"
targetNamespace="http://www.xggt.org/xmif/example-buildingPointM"
elementFormDefault="qualified" attributeFormDefault="unqualified">
  <xs:import namespace="http://www.xggt.org/xmif"
  schemaLocation="../../xmif/xmifTypes4.xsd"/>
  <xs:import namespace="http://www.xggt.org/xmif"
  schemaLocation="../../xmif/xmifMeta4.xsd"/>
  <xs:annotation>
    <xs:appinfo>
      <xsm:import schemaLocation="xmifMeta4.xsm.xsd"/>
      <xsm:sourceSchema schemaLocation="xshpBuildingPointM.xsd"/>
    </xs:appinfo>
  </xs:annotation>
  <xs:element name="Building" substitutionGroup="xmif:_Feature"
  xsm:source="fr:Building">
    <xs:complexType>
      <xs:sequence>
         <xs:sequence>
          <xs:element name="name" type="xs:string"
          xsm:source="fr:name"/>
          <xs:element name="stories" type="xs:integer"
          xsm:source="fr:stories"/>
          <xs:element name="usage" type="xs:string"
          xsm:source="fr:usage"/>
          <xs:element name="height" type="xs:double"
          xsm:source="fr:Shape//xshp:measure"/>
        </xs:sequence>
        <xs:element name="Geometry" type="xmif:GeometryPropertyType"
        xsm:source="fr:Shape"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

Figure 6.18 Schema mapping `xmifBuildingPoint.xsm.xsd`

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet xmlns:fn="http://www.w3.org/2005/xpath-functions"
xmlns:fr="http://www.xggt.org/xshp/example-buildingPointM"
xmlns:to="http://www.xggt.org/xmif/example-buildingPointM"
xmlns:xdt="http://www.w3.org/2005/xpath-datatypes"
xmlns:xmif="http://www.xggt.org/xmif"
xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:xshp="http://www.xggt.org/xshp"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
xmlns:xsm="http://www.xggt.org/mapping" version="2.0">
  <xsl:import href="../xmifMeta4.xsm.xslt"/>
  <xsl:output method="xml" version="1.0" encoding="UTF-8"
  indent="yes"/>
  <xsl:template match="fr:Building" mode="xmif:_Feature">
    <xsl:element name="to:Building">
      <xsl:for-each select="fr:name">
        <xsl:element name="to:name">
          <xsl:value-of select="."/>
        </xsl:element>
      </xsl:for-each>
      <xsl:for-each select="fr:stories">
        <xsl:element name="to:stories">
          <xsl:value-of select="."/>
        </xsl:element>
      </xsl:for-each>
      <xsl:for-each select="fr:usage">
        <xsl:element name="to:usage">
          <xsl:value-of select="."/>
        </xsl:element>
      </xsl:for-each>
      <xsl:for-each select="fr:Shape//xshp:measure">
        <xsl:element name="to:height">
          <xsl:value-of select="."/>
        </xsl:element>
      </xsl:for-each>
      <xsl:for-each select="fr:Shape">
        <xsl:element name="to:Geometry">
          <xsl:call-template name="xmif:GeometryPropertyType"/>
      </xsl:element>
      </xsl:for-each>
    </xsl:element>
  </xsl:template>
</xsl:stylesheet>
```

Figure 6.19 The generated XSLT stylesheet `xmifBuildingPoint.xsm.xslt`

The second example is a more complicated case of a one-to-many transformation. Here, MultiPoint-typed `BuildingComplex` features of XSHP are to be transformed to XMIF format, which does not support MultiPoint geometric data type. This example shows one of the possible ways of translation, in which one `BuildingComplex` in the source data is transformed to several `Building`-s in the output data. Each point in a

source MultiPoint `BuildingComplex` corresponds to a `Building` in the target data. Figure 6.20 shows the schema (`xshpBuildComplexMultiPoint.xsd`) of the source XSHP file. The imported schema defining XSHP geometric data types (`xshpTypes3.xsd`) is omitted for brevity. Figure 6.21 shows the schema mapping `xmifBuildComplexPoint.xsm.xsd` from `xshpBuildComplexMultiPoint.xsd` to `xmifBuildComplexPoint.xsd`. The imported schema mapping has been omitted for brevity. Figure 6.22 shows the generated XSLT `xmifBuildComplexPoint.xsm.xslt`. Using conventional black-box translators, such translation is either hard coded or not implemented. With the schema mapping method, various ways of dealing with such cases can be easily implemented. For instance, the centre point of the MultiPoint geometry of a source feature may be used to represent a `Building` in the target data. That can be done by providing a method of deriving the centre of a set of points and incorporating it into the schema mapping using the `xsm:method` attribute.

```
<xs:schema xmlns:xshp="http://www.xggt.org/xshp"
xmlns:fr="http://www.xggt.org/xshp/example-
buildingComplexMultiPoint"
xmlns:xs="http://www.w3.org/2001/XMLSchema"
targetNamespace="http://www.xggt.org/xshp/example-
buildingComplexMultiPoint" elementFormDefault="qualified"
attributeFormDefault="unqualified">
  <xs:import namespace=http://www.xggt.org/xshp
  schemaLocation="../../xshp/xshpTypes3.xsd"/>
  <xs:element name="BuildingComplex"
  substitutionGroup="xshp:_Feature">
    <xs:complexType>
      <xs:sequence>
       <xs:sequence>
         <xs:element name="name" type="xs:string"/>
         <xs:element name="usage" type="xs:string"/>
       </xs:sequence>
       <xs:element name="Shape"
       type="xshp:MultiPointPropertyType"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

Figure 6.20 Schema `xshpBuildComplexMultiPoint.xsd`

```
<xs:schema xmlns:xmif="http://www.xggt.org/xmif"
xmlns:xshp="http://www.xggt.org/xshp"
xmlns:xsm="http://www.xggt.org/mapping"
xmlns:to="http://www.xggt.org/xmif/example-
buildingComplexMultiPoint"
xmlns:fr="http://www.xggt.org/xshp/example-
buildingComplexMultiPoint"
xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
targetNamespace="http://www.xggt.org/xmif/example-
buildingComplexMultiPoint" elementFormDefault="qualified"
attributeFormDefault="unqualified">
  <xs:annotation>
    <xs:appinfo>
      <xsm:import schemaLocation="../xmifMeta4.xsm.xsd"/>
      <xsm:sourceSchema
      schemaLocation="xshpBuildComplexMultiPoint.xsd"/>
      <xsl:import href="../xmifFromXshpHelper.xslt"/>
    </xs:appinfo>
  </xs:annotation>
  <xs:import namespace="http://www.xggt.org/xmif"
  schemaLocation="../../xmif/xmifTypes4.xsd"/>
  <xs:import namespace="http://www.xggt.org/xmif"
  schemaLocation="../../xmif/xmifMeta4.xsd"/>
  <xs:element name="Building" substitutionGroup="xmif:_Feature"
  xsm:source="fr:BuildingComplex//xshp:componentPoints/xshp:Coordi
  nate2d">
    <xs:complexType>
      <xs:sequence>
       <xs:sequence>
         <xs:element name="name" type="xs:string"
         xsm:source="./ancestor::fr:BuildingComplex/fr:name"/>
         <xs:element name="usage" type="xs:string"
         xsm:source="./ancestor::fr:BuildingComplex/fr:usage"/>
          <xs:element name="part" type="xs:short"
         xsm:source="position()"/>
       </xs:sequence>
       <xs:element name="Geometry"
       type="xmif:GeometryPropertyType" xsm:source="."
       xsm:method="xmif:Point_from_xshp_Coordinate2d"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

Figure 6.21 Schema mapping `xmifBuildComplexPoint.xsm.xsd`

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet xmlns:fn="http://www.w3.org/2005/xpath-functions"
xmlns:fr="http://www.xggt.org/xshp/example-
buildingComplexMultiPoint"
xmlns:to="http://www.xggt.org/xmif/example-
buildingComplexMultiPoint"
xmlns:xdt="http://www.w3.org/2005/xpath-datatypes"
xmlns:xmif="http://www.xggt.org/xmif"
xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:xshp="http://www.xggt.org/xshp"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
xmlns:xsm="http://www.xggt.org/mapping" version="2.0">
  <xsl:import href="../xmifFromXshpHelper.xslt"/>
  <xsl:import href="../xmifMeta4.xsm.xslt"/>
  <xsl:output method="xml" version="1.0" encoding="UTF-8"
  indent="yes"/>
  <xsl:template
  match="fr:BuildingComplex//xshp:componentPoints/xshp:Coordinate2
  d" mode="xmif:_Feature">
    <xsl:element name="to:Building">
      <xsl:for-each
      select="./ancestor::fr:BuildingComplex/fr:name">
        <xsl:element name="to:name">
          <xsl:value-of select="."/>
        </xsl:element>
      </xsl:for-each>
      <xsl:for-each
      select="./ancestor::fr:BuildingComplex/fr:usage">
        <xsl:element name="to:usage">
          <xsl:value-of select="."/>
        </xsl:element>
      </xsl:for-each>
      <xsl:for-each select="position()">
        <xsl:element name="to:part">
          <xsl:value-of select="."/>
        </xsl:element>
      </xsl:for-each>
      <xsl:for-each select=".">
        <xsl:element name="to:Geometry">
          <xsl:call-template
          name="xmif:Point_from_xshp_Coordinate2d"/>
        </xsl:element>
      </xsl:for-each>
    </xsl:element>
  </xsl:template>
</xsl:stylesheet>
```

Figure 6.22 The generated XSLT stylesheet `xmifBuildComplexPoint.xsm.xslt`

From the examples, the following observations can be made. First, with the source and target schemas well defined, the translation process becomes conceptually clear. The developer only deals with high-level concepts. The details of data accessing are hidden. Secondly, developing custom translations has been made easier using

schema mapping as the coding effort needed is much less than that when using a procedural language.

6.5 Comparison with Related Work

There have been works by others on high-level methods of specifying XML data transformation. To compare this method with the methods of others, a sample problem of transforming a flat structure into a recursively nested structure is used as the case study. The problem is presented by W3C XML Query Work Group as case study PARTS (Chamberlin et al. 2001). As will be seen, it is basically the same as the example of transforming `flat.xml` to `tree.xml` presented earlier in this chapter.

Tang and Tompa (2001) proposed a method inspired by syntax-driven translation (SDT) (Wirth 1996) and tree transformation (TT) grammar (Keller et al. 1984). In their method, a specification consists of three components: a pair of SynTree, a set of Boolean conditions and a set of mapping rules. A SynTree is derived from a DTD (Document Type Definition) by applying production rules on non-terminals. Boolean conditions are expressed using XPath expressions. Each mapping rule consists of an operator, a left hand side of nodes from the input SynTree and a right hand side of nodes from the output SynTree. The sample specification of the PARTS case using their method is copied in Figure 6.23. Two operators were introduced, namely *copy* and *aggregate*, of which the latter does not appear in the sample. No precise meaning was defined for the operators. Although illustrative explanations on how to translate two sample specifications to XSLT were given, no algorithms and information about implementation were provided in their paper.

```
<!DOCTYPE partlist (
    <!ELEMENT partlist (part*)>
    <!ELEMENT part EMPTY>
    <!ATTLIST part
          partid  CDATA  #REQUIRED
          partof  CDATA  #IMPLIED
          name    CDATA  #REQUIRED>
)>
```

(a) partlist.dtd

```
<!DOCTYPE parttree (
    <!ELEMENT parttree (part*)>
    <!ELEMENT part (part*)>
    <!ATTLIST part
          partid  CDATA  #REQUIRED
          name    CDATA  #REQUIRED>
)>
```

(b) parttree.dtd

Input SynTree: i:partlist< i:part*< i:@partid i:@partof? i:@name >>
Output SynTree: o:parttree< o:part[C1]*< o:@partid o:@partof o:part[C2]*<> >>
Boolean Conditions: C1: not(i:@partof)
C2: i:@partof = ../o:@partid
Mapping Rules: *copy*: i:partlist → o:parttree
copy: i:part → o:part[1] o:part[2]

(c) transformation specification from partlist.dtd to parttree.dtd

Figure 6.23 Transformation specification of the PARTS case using method of Tang and Tompa (2001)

```
<!DOCTYPE partlist (
   <!ELEMENT partlist (part*)>
   <!ELEMENT part (name, partof*)>
      <!ATTLIST part
      id CDATA #REQUIRED>
      <!ELEMENT name (#PCDATA)>
      <!ELEMENT partof (#PCDATA)>
)>
```

(a) partlist.dtd

```
<!DOCTYPE parttree (
   <!ELEMENT parttree (part*)>
   <!ELEMENT part (name, part*)>
      <!ATTLIST part
      id CDATA #REQUIRED>
      <!ELEMENT name (#PCDATA)>
)>
```

(b) parttree.dtd

(PARTLIST, /) → PARTTREE
(PARTTREE, ./partlist) → parttree(MAIN_PART)
(MAIN_PART, ./part[not partof]) → part(ID, NAME, SUB_PART)
(ID, ./@id) → @id()
(NAME, ./name) → name()
(SUB_PART, ../part[partof/text()= = $in/@id]) → part(ID, NAME, SUB_PART)

(c) transformation specification from partlist.dtd to parttree.dtd

Figure 6.24 Transformation specification of the PARTS case using method of Pankowski (2003) (note: in Pankowski (2003), the DTD-s have minor differences)

```
<?xml version="1.0" encoding="UTF-
8"?>
<xs:schema
xmlns:xs="http://www.w3.org/2001/XML
Schema"
xmlns:pl="http://www.xggt.org/mappin
g/example-partlist"
targetNamespace="http://www.xggt.org
/mapping/example-partlist">

<xs:element name="partlist">
 <xs:complexType>
   <xs:sequence>
    <xs:element ref="pl:part"
    maxOccurs="unbounded"/>
   </xs:sequence>
  </xs:complexType>
 </xs:element>

 <xs:element name="part">
  <xs:complexType>
   <xs:attribute name="partid"
   type="xs:integer"/>
   <xs:attribute name="partof"
   type="xs:integer"/>
   <xs:attribute name="name"
   type="xs:string"/>
  </xs:complexType>
 </xs:element>

</xs:schema>
```

(a) partlist.xsd

```
<?xml version="1.0" encoding="UTF-
8"?>
<xs:schema
xmlns:xs="http://www.w3.org/2001/XML
Schema"
xmlns:pt="http://www.xggt.org/mappin
g/example-parttree"
targetNamespace="http://www.xggt.org
/mapping/example-parttree">
 <xs:element name="parttree">
  <xs:complexType>
   <xs:sequence>
    <xs:element ref="pt:part"
    maxOccurs="unbounded"/>
   </xs:sequence>
  </xs:complexType>
 </xs:element>
 <xs:element name="part">
  <xs:complexType>
   <xs:sequence>
    <xs:element ref="pt:part"
    minOccurs="0"
    maxOccurs="unbounded"/>
   </xs:sequence>
   <xs:attribute name="partid"
   type="xs:integer"/>
   <xs:attribute name="name"
   type="xs:string"/>
  </xs:complexType>
 </xs:element>
</xs:schema>
```

(b) parttree.xsd

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:xsm="http://www.xggt.org/mapping"
xmlns:pt="http://www.xggt.org/mapping/example-parttree"
xmlns:pl="http://www.xggt.org/mapping/example-partlist"
targetNamespace="http://www.xggt.org/mapping/example-parttree">
  <xs:annotation>
    <xs:appinfo>
      <xsm:sourceSchema schemaLocation="partlist.xsd"/>
      <xsm:root name="pt:parttree"/>
    </xs:appinfo>
  </xs:annotation>
  <xs:element name="parttree" xsm:source="pl:partlist">
    <xs:complexType>
      <xs:sequence>
         <xs:element ref="pt:part" maxOccurs="unbounded" xsm:source="pl:part
         (not(@partof))"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:element name="part" xsm:source="">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="pt:part" minOccurs="0" maxOccurs="unbounded"
        xsm:source="../pl:part (@partof=current()/@partid)"/>
      </xs:sequence>
      <xs:attribute name="partid" type="xs:integer" xsm:source="@partid"/>
      <xs:attribute name="name" type="xs:string" xsm:source="@name"/>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

(c) transformation specification from partlist.xsd to parttree.xsd

Figure 6.25 Transformation specification of the PARTS case using method of this study

Pankowski (2003) proposed a method of specifying transformation of XML data based on unranked tree transduction. An intermediate data/document definition is necessary to assist in specifying transformation, which is specified as a sequence of production rules, each relating source data definitions to the intermediate data definitions and on to the target data definitions. The sample specification of the PARTS case given by Pankowski (2003) is copied in Figure 6.24. In the figure, symbols of upper case letters are intermediate data definition symbols. Symbols in lower case letters at the left hand side of production rules are source data definition symbols, while those at the right hand side are target data definition symbols. Algorithms for translating transformation specifications to XSLT are provided. One example of transformation specification was given, which is the one copied here. No implementation information was provided. As can be seen, the need of an intermediate data definition is a critical limit of Pankowski's method, which makes transformation specification less intuitive. Moreover, that makes automatic derivation of transformation specifications a harder task.

There have also been commercial products for specifying XML data transformation. One of the well-known products is the above-mentioned MapForce® from Altova GmbH (Altova 2005). MapForce provides a graphical interface for specifying transformation at schema level and is able to generate codes in several programming languages. However, for the PARTS case, this author cannot find a way to do the job with its latest release. It seems to this author that MapForce® cannot deal with transforming data that involves recursive definitions as the PARTS study case does. In addition, MapForce® does not provide a textual mapping specification language, which means that human operators cannot edit the intermediate result of schema mapping. Nor is there a mechanism of schema mapping modularity and reuse, which

is likely due to the lack of a textual counterpart to the graphical schema mapping method. The same observations apply to Stylus Studio®, another well-known XML workbench from DataDirect Technologies, which provides some function of specifying XML transformation through GUI.

For comparison, the transformation specification for the PARTS case using the method of this study is presented in Figure 6.25. Note that DTDs have been replaced by schemas. Despite the verbose impression of this specification due to the use of XML syntax, the following positive observations can be made. Standard schema annotation is used instead of proprietary mapping syntax. No intermediate data definitions are needed. Source objects are mapped to target objects directly, which makes a schema mapping conceptually clear and easy to set. In addition, global *elements* (and *complexType*-s, which cannot be seen in this examples) and *references* to them are treated conceptually "right"; that is, global *elements* and *complexTypes* are treated as data type/class definitions while *references* to *elements* and uses of *complexTypes* are treated as instantiation of types/classes. That makes it easy in this method to deal with recursive definitions. Further, the method used by this study to transform specifications has been tested with quite a number of cases. In addition to the examples shown (and not shown) in this chapter, the same method will be used to specify meta-schema mapping in the next chapter.

6.6 Summary and Discussion

Structural transformation is at the core of the data translation problem. In this chapter, a schema mapping specification method has been proposed. It is based on WXS. Schema mapping is specified by annotating a target schema with assertions of correspondence from a source schema. It enables high-level specification of geodata

transformation without getting involved in data accessing details. Imperative programming is thus not needed for common data restructuring tasks.

Although the proposed schema mapping method has not been developed to fully support WXS, its implemented subset shows that the idea of generating a XSLT stylesheet from schema mapping is beneficial. Algorithms for translating schema mapping specifications into XSLT stylesheets have been developed and implemented. The examples presented show that the schema mapping specification method is expressive and declarative.

Although the proposed schema mapping method aims at facilitating geodata translation, it can be used as a general method of specifying XML data transformation at schema level. The proposed textual schema mapping method also provides a basis for designing graphical user interfaces for specifying schema mapping.

It is worth noting that the schema mapping specification method is by no means a replacement for XSLT or imperative programming languages, just as XSLT is not a replacement for imperative programming languages for transforming XML data. The objective is to make common tasks easier; in particular, making reconciliation of schematic differences easier.

Finally, current XSLT implantations require in-memory tree representation of XML data. That means large memory consumption in the transformation process. For large datasets, the memory demand can be prohibitive. Optimization of XSLT transformation is needed. Indeed, such optimization is possible and now under research (Kay 2001).

Chapter 7 Generating Schema Mapping Generators from Meta-Schema Mappings

7.1 Introduction

In the previous chapter, a schema mapping specification method has been proposed. The method, which can be considered as a sort of declarative schema mapping language, makes common tasks of data transformation easy to specify. Schema mapping is useful when a target schema has been defined and data transformation needs to produce data according to the target schema. In the case of data translation from one format to another, a default target schema often needs to be obtained by first translating the source schema and then revising it to better suit the target application. In other words, schema translation needs to be addressed in data translation as it has always been.

When writing geodata translators in the conventional way of brute force programming, everything is manually coded and so is part of the schema translator. However, in our approach simply translating a source schema to a target schema without providing the associated schema mapping between them is not sufficient. Schema mapping would have to be specified by human operators when a target schema is provided, but without the mapping from its source. Hand-coding schema translators that can produce schema mappings together with target schemas would be an added burden. So, it turns out that what was once desired, i.e. automating schema translator generation, now becomes a requirement. Further, the (to be generated) schema translator needs to be an enhanced one that can generate schema mappings in addition to translating schemas. These are the concerns of this chapter.

The method for specifying meta-schemas is proposed in Section 7.2. This is followed by a brief explanation on re-using the schema mapping specification method for meta-schema mapping specification in Section 7.3. Then, the core issue of deriving schema mapping from meta-schema mapping and schema instances is addressed in Section 7.4, together with the discussion on generating schema mapping generators. A case study is presented in Section 7.5. This work is compared with two related works in Section 7.6. Finally, Section 7.7 gives a summary and some discussion.

7.2 Specifying Meta-Schemas

In the previous chapter, the translation of one XML data instance file conforming to a data schema to another XML data instance file conforming to a second data schema was discussed. It was taken for granted that both schemas were WXS schemas. Given that both schemas are WXS schemas, it seems ridiculous to talk about schema translation. Hence, an explanation is needed.

As is known, a data schema specifies the structure of a set of datasets, possibly with semantic constraints. It thus can be used to check if a data file is a valid instance of it in the same sense as checking if a sequence of symbols is a valid sentence of a language specified by a grammar. A data schema can be seen as a kind of data instance with respect to a meta-schema. In other words, a meta-schema can be used to specify a set of allowed data schemas in the same way as using a data schema to specify a set of data instances that are allowed, as illustrated in Figure 7.1. In fact, this has already been done. When a WXS schema is written down, it is necessary to know if it is valid according to the WXS meta-schema. A WXS schema processor needs to be able to do that. For example, the schema `xshpBuildingPointM.xsd`

given in the previous chapter has been validated against the WXS meta-schema using XMLSpy® from Altova GmbH.

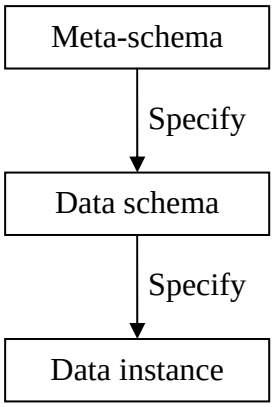


Figure 7.1 Meta-schema, schema and data instance

However, the fact that the schema `xshpBuildingPointM.xsd` is a valid WXS schema does not mean that it is a valid schema for XSHP data, or a valid XSHP schema. The set of all valid XSHP schemas is a subset of all valid WXS schemas. That means there is a meta-schema associated with the XML-based file format XSHP introduced in Chapter 4, which needs to be formally specified and then used to check if a WXS schema is a valid schema for some XSHP data file.

7.2.1 Specifying Meta-Schemas using WXS

The approach of specifying meta-schemas adopted in this study is simply to take a set of WXS schemas as ordinary XML documents and use WXS to specify a schema for them, although the resulting schema is in fact a meta-schema. Thus, the resulting meta-schema uses a subset of vocabulary of WXS in a restricted way. With this method, the meta-schemas of XSHP and XMIF have been defined and presented in Appendix D and E, respectively. (In fact, what is given in Appendix E is the XMIF

meta-schema mapping from XSHP meta-schema. See next section for explanation.) Here, XSHP meta-schema is taken as the example to illustrate this method.

In Shape/XSHP data file format, users are allowed to define a sort of geo-relational schema, in which a single geographic feature class can be defined for a geo-relational table. For a geographic feature class, a sequence of non-geometric attributes and a single geometric attribute can be defined. The data type of the geometric attribute should be among those that have been defined in `xshpTypes3.xsd`. (Strictly speaking, the literal or simple data types available in SHAPE/XSHP format also need to be specified, but this has not been done in `xshp4_1.xsd`. Several WXS built-in simple types roughly equivalent to SHAPE simple types are used instead.) Users are not allowed to define their own data types, either non-geometric or geometric. Users are allowed to choose a name for non-geometric attributes but not allowed to choose a name for the geometric attribute.

To define a meta-schema for XSHP data schemas is to specify the above described meta-model formally so as to check if an ordinary WXS schema is a schema for an XSHP file, as well as to facilitate schema translation. How this is done using WXS will now be discussed in detail. The meta-schema, `xshp4_1.xsd`, for XSHP files is presented in Appendix D. Fragments of the meta-schema will be cited to help illustration.

A namespace “http://www.xggt.org/xspxshp” is introduced to represent the XSHP meta-schema vocabulary. “xspxshp” will be used as its prefix in this writing and “XSPXSHP” will be used as an adjective meaning “related to XSHP meta schema”. Similarly, “xspxmif” and XSPXMIF will be used in the same way.

It is interesting to note that the elements defined in *xspxshp* namespace share the same names of those of WXS schema elements. For example, the `(xspxshp:)schema` element corresponds to `(xs:)schema` element, while the `(xspxshp:)element` element corresponds to `(xs:)element` element. Hence, an XSPXSHP schema will look exactly the same as a WXS schema if namespaces are not taken into account. That also means if "xspxshp" is changed to become the namespace prefix of "http://www.w3.org/2001/XMLSchema", then an XSPXSHP schema will be a valid WXS schema. Therefore, to check the validity of an XSPXSHP schema, the "xspshp" prefix can be used to represent the "http://www.xggt.org/xspxshp" namespace and taken as an ordinary XML data file and validated against the XSHP meta-schema (i.e. `xshp4_1.xsd`). To use a XSPXSHP schema to validate a XSHP data instance file, the "xspxshp" prefix can be used to represent the "http://www.w3.org/2001/XMLSchema" namespace, which can thus be used as an ordinary WXS schema. In implementation, two copies of a XSPXSHP schema are used, which are identical except that prefix "xspxshp" refers to "http://www.xggt.org/xspxshp" in one copy while it refers to "http://www.w3.org/2001/XMLSchema" in the other.

```
<xs:element name="schema">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="xspxshp:import"/>
      <xs:element ref="xspxshp:annotation" minOccurs="0"
      maxOccurs="unbounded"/>
      <xs:element name="element" type="xspxshp:FeatureClass"/>
    <xs:/sequence>
    <xs:attribute name="attributeFormDefault"
    type="xspxshp:formEnum" default="unqualified"/>
    <xs:attribute name="elementFormDefault"
    type="xspxshp:formEnum" default="unqualified"/>
    <xs:attribute name="targetNamespace" type="xs:anyURI"/>
    <xs:attribute name="version" type="xs:token"/>
    <xs:anyAttribute namespace="##other" processContents="skip"/>
   </xs:complexType>
</xs:element>
```

Figure 7.2 Definition of the *schema* element of XSHP meta-schema

Figure 7.2 gives the definition of the `schema` element of XSHP meta-schema. It basically states an XSPXSHP schema consists of an `xspxshp:import` and a `xspxshp:FeatureClass` definition. The `xspxshp:import` definition introduces the geometric data types defined elsewhere. The focus will now be on `xspxshp:FeatureClass` definition, which is shown Figure 7.3.

```
<xs:complexType name="FeatureClass">
 <xs:sequence>
  <xs:element ref="xspxshp:annotation" minOccurs="0"
  maxOccurs="unbounded"/>
  <xs:element name="complexType">
   <xs:complexType>
    <xs:sequence>
     <xs:element ref="xspxshp:annotation" minOccurs="0"
     maxOccurs="unbounded"/>
     <xs:element name="sequence">
      <xs:complexType>
       <xs:sequence>
        <xs:element ref="xspxshp:annotation" minOccurs="0"
        maxOccurs="unbounded"/>
        <xs:element name="sequence" type="xspxshp:FieldSequence"/>
        <xs:element name="element" type="xspxshp:G_Field"/>
       </xs:sequence>
      </xs:complexType>
     </xs:element>
    </xs:sequence>
   </xs:complexType>
  </xs:element>
 </xs:sequence>
 <xs:attribute name="name" type="xs:NCName"/>
 <xs:attribute name="substitutionGroup" type="xs:QName"
 fixed="xshp:_Feature"/>
</xs:complexType>
```

Figure 7.3 Definition of the *FeatureClass* complexType of XSHP meta-schema

The `xspxshp:FeatureClass` *complexType* specifies that a feature class definition includes a sequence of non-geometric field definitions and a geometric field definition. As shown in Figure 7.4, a non-geometric field definition is to be specified as an `xspxshp:element` of the type of `xspshp:Field` and the element can have a (user-given) name and should be of type as enumerated in the `xspxshp:SimpleTypeEnum`. A geometric field definition is to be specified as an `xspxshp:element` of type `xspxshp:G_Field` and should have a fixed name of "Shape" but can be of any type as enumerated in the `xspxshp:ShapeTypeEnum`.

```
<xs:complexType name="FieldSequence">
  <xs:sequence>
    <xs:choice minOccurs="0" maxOccurs="unbounded">
      <xs:element ref="xspxshp:annotation"/>
      <xs:element name="element" type="xspxshp:Field"/>
    </xs:choice>
  </xs:sequence>
</xs:complexType>

<xs:complexType name="Field">
  <xs:attribute name="name" type="xs:NCName"/>
  <xs:attribute name="type" type="xspxshp:SimpleTypeEnum"/>
  <xs:anyAttribute namespace="##other" processContents="skip"/>
 </xs:complexType>

<xs:complexType name="G_Field">
  <xs:attribute name="name" type="xs:NCName" fixed="Shape"/>
  <xs:attribute name="type" type="xspxshp:ShapeTypeEnum"/>
  <xs:anyAttribute namespace="##other" processContents="skip"/>
</xs:complexType>
```

Figure 7.4 Definitions of the *FieldSequence*, *Field and G_Field* complexType-s of XSHP meta-schema

A sample XSPXSHP schema is shown in Figure 7.5. As can be seen, if prefix *xspxshp* is changed to represent "http://www.w3.org/2001/XMLSchema"; it thus becomes a valid WXS schema.

```
<?xml version="1.0" encoding="UTF-8"?>
<xspxshp:schema xmlns="http://www.w3.org/2001/XMLSchema"
xmlns:xspxshp="http://www.xggt.org/xspxshp"
xmlns:xshp="http://www.xggt.org/xshp"
targetNamespace="http://www.xggt.org/example"
elementFormDefault="qualified" attributeFormDefault="unqualified"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.xggt.org/xspxshp xshp4_1.xsd">

 <xspxshp:import namespace="http://www.xggt.org/xshp"
 schemaLocation="xshpTypes3.xsd"/>

 <xspxshp:element name="Lake" substitutionGroup="xshp:_Feature">
  <xspxshp:complexType>
   <xspxshp:sequence>
    <xspxshp:sequence>
     <xspxshp:element name="name" type="string"/>
     <xspxshp:element name="area" type="decimal"/>
     <xspxshp:element name="surf_elev" type="decimal"/>
     <xspxshp:element name="depth" type="decimal"/>
    </xspxshp:sequence>
    <xspxshp:element name="Shape" type="xshp:PolygonPropertyType"/>
   </xspxshp:sequence>
  </xspxshp:complexType>
 </xspxshp:element>

</xspxshp:schema>
```

Figure 7.5 A sample XSPXSHP schema (`xshpLakePolygon.xsd.xml`)

7.2.2 Other Methods of Specifying Meta-Schemas using WXS

There are other ways of defining meta-schemas using WXS. Provost (2002) suggested using the *redefine* construct in WXS upon the WXS meta-schema. However, given the complexity of WXS, that is a very challenging method because one would have to be a master of WXS. Moreover, the resulting meta-schema would look even more complicated than the original WXS meta-schema no matter how simple in fact it was.

WXS provides a *restriction* construct, which, in conjunction with other constructs such as *abstract/substitutionGroup*, provides a mechanism of defining a restricted version of another schema. That seems to be a means of implementing two-level (or even multiple-level) modelling, in which a meta-schema may be defined as an

abstract WXS schema consisting of abstract elements or types, while concrete schemas define elements and types that are substitutions or extensions/restrictions of them. This method is heavily used in defining Geographical Markup Language (OGC 2002). However, that turns out not to be the same as what is needed as a method of specifying a meta-schema.

On the one hand, not enough constraining mechanisms are available in this approach to define a genuine meta-schema. For example, if one wants to express the constraint that a non-geometric attribute of XSHP can have a user-given name but it can only be of a simple type that is among pre-defined simple types, then first of all a complex type for each possible simple type needs to be defined and specified to block restriction and extension. Then an abstract element of that complex type has to be defined to serve as a group to be substituted. In an application (i.e. user-defined) schema, an attribute (in the sense of XSHP) has to be defined as a substitution to the abstract element. That not only makes an (application) schema complicated and look very odd, but more importantly it does not prevent a user from defining non-predefined types and freely using the remaining WXS modelling constructs.

On the other hand, complicated schemas making intensive use of *restriction, abstract element/complexType* and *substitutionGroup* cannot be properly processed with current WXS processors due to the already mentioned WXS implementation inconsistencies. As an example, a verified example of GML 2.1.2 (the last version of GML 2, which should be considered as a stable specification) presented in GML 2.1.2 specification (OGC 2002) cannot pass the validation of the latest version of the widely used XMLSpy software tool as tested in this study.

Therefore, compared with the methods discussed in this subsection, the approach adopted to meta-schema specification in this study is favoured, for it is both generic and implementation-viable.

7.3 Generating Schema Translators from Meta-Schema Mappings

With meta-schemas defined as WXS schemas, meta-schema mappings from a source meta-schema to a target meta-schema can be defined using the schema mapping method proposed in the last chapter. A parallel between schema mapping and meta-schema mapping is drawn in Figure 7.6. From a meta-schema mapping, an XSLT stylesheet can be derived, which can translate a source schema to a target schema. That means, for a certain pair of source meta-schema and target meta-schema, to develop a schema translator from the source to the target is to define the mapping between them using the high-level mapping language and have the translator generated from the mapping. Thus it can be said that developing schema translators has been made much easier. It is also convenient to specify more than one meta-schema mapping for a specific pair of source meta-schema and target meta-schema so as to have different ways of model translation for different uses.

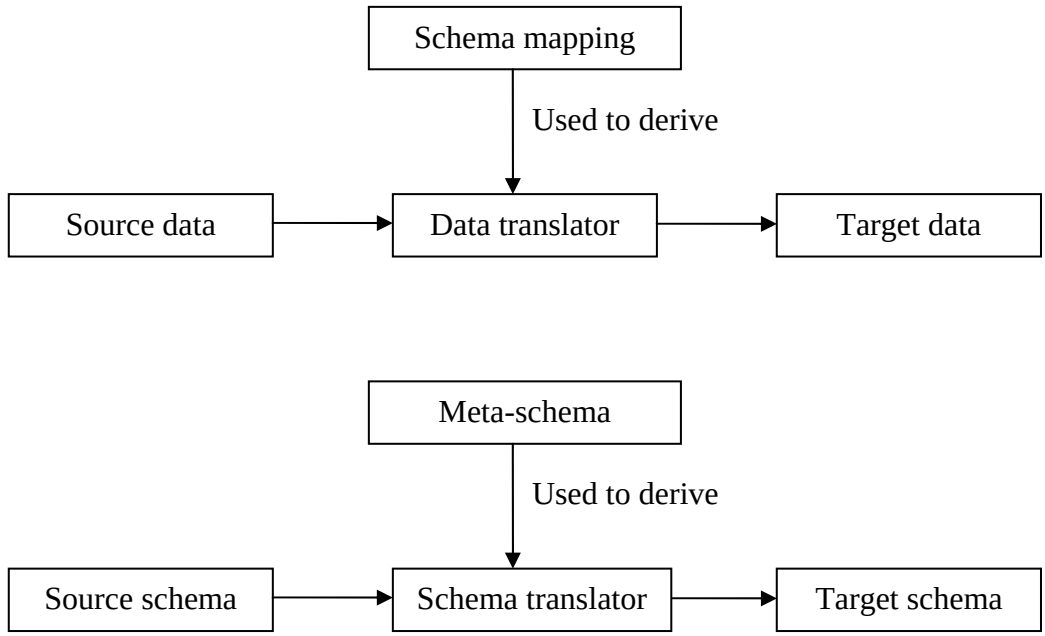


Figure 7.6 The parallel relationship of schema mapping and meta-schema mapping (For conceptual clarity, the schema mapping processor is omitted.)

A meta-schema mapping from XSHP meta-schema to XMIF meta-schema has been defined and is presented in Appendix E, which, not surprisingly, is the XMIF meta-schema annotated with mapping information. The "clean" version of XMIF meta-schema is not duplicated. It has been translated by the schema-mapping processor to an XSLT stylesheet. The resulting stylesheet is in fact an XSHP-to-XMIF schema translator that can transform an XSPXSHP schema to an XSPXMIF schema. The process of translating a meta-schema mapping to a schema translator is not elaborated here, as it is almost identical to the process of translating a schema mapping to a data instance translator. The major concern is to generate schema mappings from a meta-schema mapping.

7.4 Generating Schema Mapping Generators

As has been explained, simply deriving a target schema from a source schema is not enough for our purpose. The associated schema mapping is also needed. That is,

there needs to be a means to chain meta-schema mapping to schema mapping. As mentioned above, the method proposed by this study is to enhance the (meta-) schema mapping processor so that it can generate an enhanced schema translator that can produce an associated schema mapping as well as a target schema. The idea is illustrated in Figure 7.7.

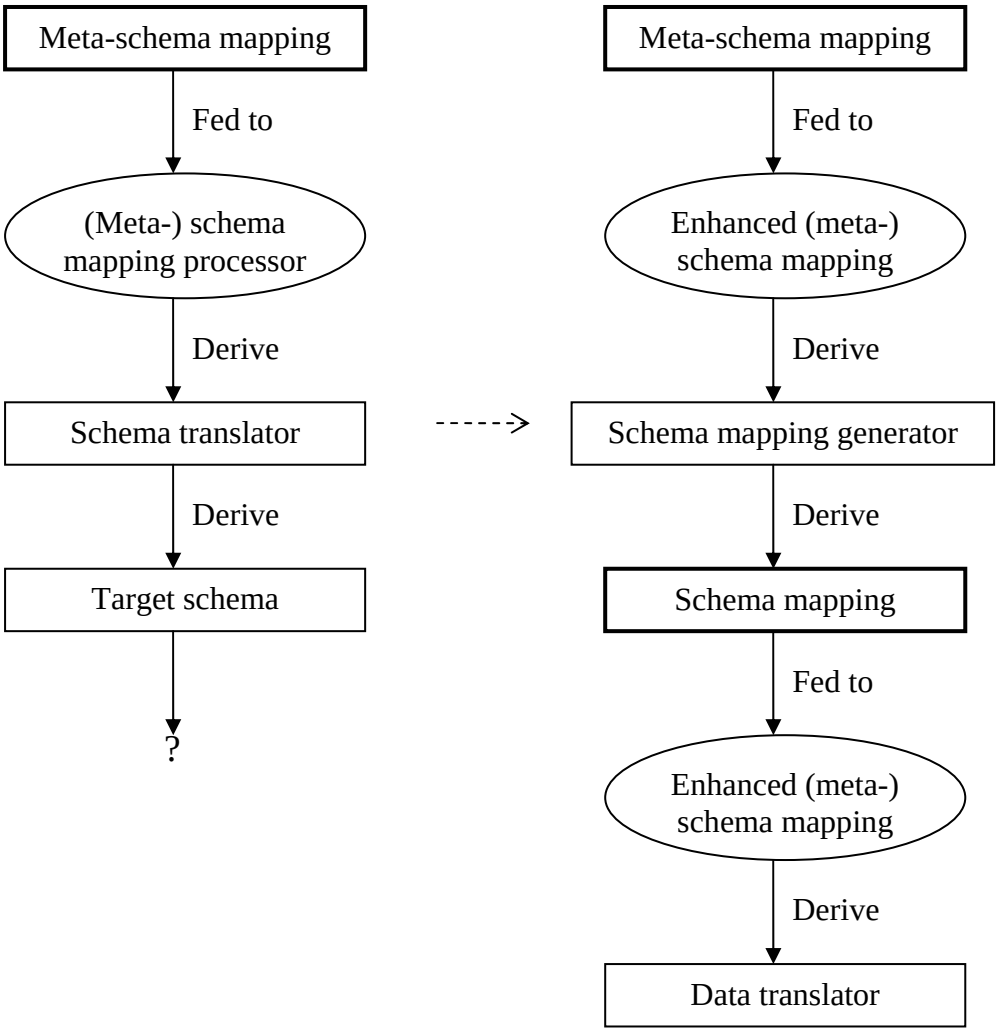


Figure 7.7 Chaining meta-schema mapping to schema mapping

As a schema mapping is just the target schema annotated with mapping information, the (meta-) schema mapping processor needs only be extended in such a way that it adds the mapping annotations to the target schema being generated. As can be seen from the last chapter, the core issue in specifying schema mapping is to specify the

correspondence between source nodes and target nodes using the *xsm:source* attribute. Thus, in the process of annotating a target schema with mapping information, the generation of *xsm:source* attribute is the core issue. A close investigation into the schema translation process reveals that the associated schema mapping can be derived. In the following, how data instance correspondences (or mappings) can be obtained from the associated schema-mapping is analysed, with an example. Then, the method is extended to obtain schema mappings from the associated meta-schema mapping.

7.4.1 Deriving Instance Mapping from Schema Mapping

In the translation process, a translator knows the source item of every output item. As usually only the result is of interest, the correspondences between source items and output items are not given, but could be if desired. An example of translation at data instance level helps illustrate this.

```
<?xml version="1.0" encoding="UTF-8"?>
<tree:root xmlns:tree="mapping:examples-itemTree"
      absolute="/root"
    context="/"
    relative="flat:root">
       <tree:item id="1"
    absolute="/root/item(1)"
    context="flat:root"
    relative="flat:item (@parent=0)">
             <tree:item id="2"
       absolute="/root/item(2)"
       context="flat:item (@parent=0)"
       relative="../flat:item (@parent=1)">
                  <tree:item id="3"
          absolute="/root/item(3)"
          context="../flat:item (@parent=1)"
          relative="../flat:item (@parent=2)"/>
                  <tree:item id="5"
          absolute="/root/item(5)"
          context="../flat:item (@parent=1)"
          relative="../flat:item (@parent=2)">
                      <tree:item id="6"
             absolute="/root/item(6)"
             context="../flat:item (@parent=2)"
             relative="../flat:item (@parent=5)"/>
                  </tree:item>
            </tree:item>
            <tree:item id="4"
       absolute="/root/item(4)"
       context="flat:item (@parent=0)"
       relative="../flat:item (@parent=1)"/>
      </tree:item>
</tree:root>
```

Figure 7.8 Instance correspondences between *flat.xml* and *tree.xml* obtained from the combination of schema mapping *flat-to-tree.xsm.xsd* and *flat.xml* (*@absolute* stands for the absolute path of the source node. *@context* stands for the contextual source node; *@relative* stands for the relative path of source node with respect to the contextual source node)

Continuing the example of translating `flat.xml` to `tree.xml` in the last chapter, Figure 7.8 shows the correspondence between output items and source items. In Figure 7.8, the *absolute* attribute gives the absolute path of the source data item that corresponds to the current output data item. The *absolute* attribute is obtained by analysing the absolute path of the current source node in the source data tree without resorting to the schema mapping. However, in specifying the correspondence at schema level, the relative source path is needed. A relative source path is the relative

path of the current source node with respect to the source node of the immediate upper level target node (called context source node here). The notion of relative source path is illustrated in Figure 7.9. In Figure 7.8, the *relative* attribute gives the relative source path. The *context* and *relative* attributes are obtained from the schema mapping, i.e. `tree.xsm.xsd`. In particular, the *relative* attribute of an output node is simply a "context-dependent" copy of the current *xsm:source* attribute specified in the schema mapping. "Context-dependent" copy means a literal copy, except that if the path expression being copied contains any context-dependent value, such as variables and context-dependent functions, then those values need to be evaluated and the result of the evaluation should be copied instead. For the current example, two instances of xsm:source attribute appear, which are cited below:

```
<xs:element ref="tree:item" … xsm:source="flat:item (@parent=0)"/>
<xs:element ref="tree:item" … xsm:source="../flat:item
(@parent=current()/@id)"/>
```

The first `xsm:source="flat:item (@parent=0)"` does not contain any context-dependent value. In the second, `xsm:source="../flat:item (@parent=current()/@id)"`, `current()/@id` is a context-dependent expression and thus needs to be evaluated.

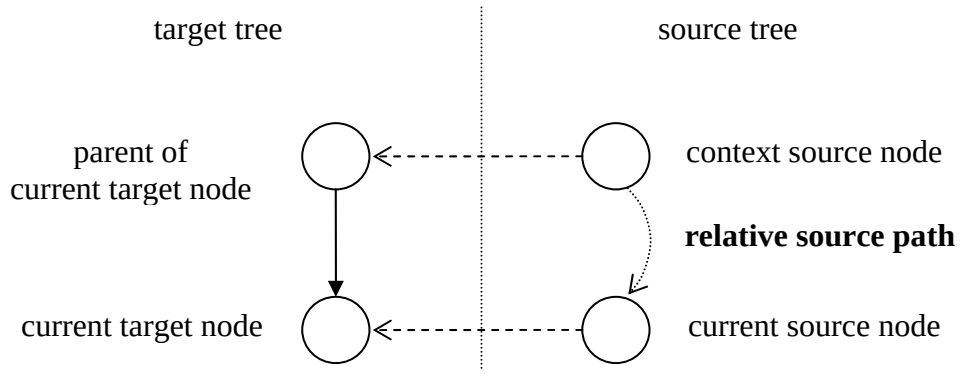


Figure 7.9 An illustration of relative source path

From this example, it can be observed that data instance level correspondences can be obtained from a combination of the associated schema mapping and the source data tree in the process of data translation. In particular, the relative path of the source node corresponding to a target node being generated is the context-dependent copy of the *xsm:source* attribute associated with the target node.

The derivation of schema-level *xsm:source* attribute from a meta-schema mapping is basically the same as the derivation of relative source paths of instance source nodes from schema-mapping discussed above. The schema-level *xsm:source* attributes need to be derived from the combination of the meta-schema mapping and a source schema. As will be seen, in this combination, the meta-level *xsm:source* attribute provides the pattern of schema-level *xsm:source* attributes while information from a source schema fills parameters in the pattern. The difference is that relative source paths in a meta-schema mapping need to be converted to schema-level paths.

7.4.2 Generating xsm:source Attributes

In a meta-schema, there are schema definition elements that do not correspond to instance level nodes. For instance, in a meta-schema, there can be a definition of a *sequence* element, which is used to help specify schemas, and does not correspond to a node appearing in instance data. That introduces meta-schema mapping items that do not correspond to schema mapping items. These are referred to as *meta-level assistance mapping*. A meta-schema mapping fragment of *xspxmif:FeatureClass* from *xspxshp:FeatureClass* is shown in Figure 7.10 to help explain. In Figure 7.10, the following meta-schema mapping items do not correspond to schema-level mapping items:

```
<element name="complexType" xsm:source="xspxshp:complexType">
```

```
<element name="sequence" xsm:source="xspxshp:sequence">
```

Such meta-level mappings need to be ignored. In contrast, the following meta-schema mapping item does correspond to schema-level mapping item and needs to be converted to schema-level mapping:

```
<element name="element" type="xspxmif:G_Field"
xsm:source="xspxshp:element"/>
```

```
<complexType name="FeatureClass"
xsm:source="xspxshp:FeatureClass">
 <sequence>
  <element ref="xspxmif:annotation" minOccurs="0"
  maxOccurs="unbounded"/>
  <element name="complexType" xsm:source="xspxshp:complexType">
   <complexType>
    <sequence>
     <element ref="xspxmif:annotation" minOccurs="0"
     maxOccurs="unbounded"/>
     <element name="sequence" xsm:source="xspxshp:sequence">
      <complexType>
       <sequence>
        <element ref="xspxmif:annotation" minOccurs="0"
        maxOccurs="unbounded"/>
        <element name="sequence" type="xspxmif:FieldSequence"
        xsm:source="xspxshp:sequence"/>
        <element name="element" type="xspxmif:G_Field"
        xsm:source="xspxshp:element"/>
       </sequence>
      </complexType>
     </element>
    </sequence>
   </complexType>
  </element>
 </sequence>
 <attribute name="name" type="NCName" xsm:source="@name"/>
 <attribute name="substitutionGroup" type="QName"
 fixed="xmif:_Feature" xsm:source=""/>
 <anyAttribute namespace="##other" processContents="skip"/>
</complexType>
```

Figure 7.10 Meta-schema mapping fragment of xspxmif:FeatureClass from xspxshp:FeatureClass

In the previous chapter, it was shown that an *xsm:source* attribute can be attached to *xs:element*, *xs:element reference*, *xs:attribute*, global *xs:complexType* and global *xs:simpleType*. Therefore, when a meta-schema definition specifies such (schema-level) objects and itself has an *xsm:source* attribute, there is a need to append an

xsm:source attribute to the schema-level object being generated. For other cases, a meta-level *xsm:source* attribute does not correspond to schema-level *xsm:source* attributes and should be ignored.

7.4.3 Deriving xsm:source Attributes of Target Global Objects

In a schema, two kinds of definition can be distinguished, namely local and global. Local definitions define local objects, including *elements, element references, attributes, and attribute references*, etc. Global definitions define global *elements* and global *types*, etc. From the definition of the *xsm:source* attribute given in the previous chapter, it is known that the *xsm:source* attribute for a global object is either an empty string or the name of the corresponding source global object, if there is one. For a local object, its *xsm:source* attribute is either an empty string or an XPath path expression of its correspondence source object. So, the algorithm for deriving *xsm:source* attributes of global objects is quite straightforward as shown in Figure 7.11.

For the target root element, *xsm:source* = the *absolute path* of its source object.
For a target global object *t*,
    If its source object *s* is global, then *xsm:source* = *name* of *s* .
    If its source object is local, then *xsm:source* = "" .
For a target local object *t*,
    If its source object *s* is local, then *xsm:source* = (result of resolving relative path with respect to its immediate upper level mapping).
    If its source object is global, then ERROR.

Figure 7.11 Algorithm for deriving xsm:source attributes of global objects

7.4.4 Deriving xsm:source Attributes of Target Local Objects

As explained earlier, the schema-level *xsm:attribute* of a target local object is a relative source path that can be derived from a combination of meta-level

xsm:attribute and the source local object being processed. In particular, each step of a relative path in meta-level *xsm:source* attribute needs to be converted to a schema-level path step. In addition, the resulting schema-level path needs to be normalized to be in a simple form. These are analysed as follows.

Converting meta-level path step to schema-level path step

As explained in Subsection 7.4.2, there are schema definition elements that do not correspond to instance level nodes in a meta-schema. This may introduce *self* stepping in schema-level relative source paths, as well as the problem of meta-level assistance mapping discussed above. For instance, the *xsm:source* attribute in the following fragment of meta-level mapping indicates a change of current path in travelling a source data tree.

```
<element name="element" type="xspxmif:G_Field"
xsm:source="xspxshp:element"/>
```

In contrast, the following fragment of meta-level mapping does not result in a change of current path in travelling a source data tree.

```
<element name="sequence" type="xspxmif:FieldSequence"
xsm:source="xspxshp:sequence"/>
```

So, a path step in a meta-level *xsm:source* attribute that does not correspond to node selection instruction at instance level does not change the path and therefore needs to be replaced by a *self* step. That is, a meta-level path step such as `xspxshp:sequence` will be replaced by `"./"` in the process of deriving schema level paths.

However, a path step that corresponds to node selection instruction at instance level will change the path. Further, a meta-level path step specifies only the node type of a step, which can be either *element* or *attribute*. The name of the *element* or *attribute* needs to be defined and is contained in the source schema being processed. Therefore,

a meta-level path step of *element* or *attribute* type will be converted to a schema-level path step and the *name* used for the node test will be defined in the schema translation process with the name of the current schema node. For instance, in the above cited mapping item, the meta-level path step `xspxshp:element` in `xsm:source="xspxshp:element"` will be converted to a schema-level path step, such as `fr:depth` and `fr:Shape`, depending on the schema node being processed.

Simplifying path expression

The resulting paths obtained from the above processing will be valid schema-level paths, but may not appear as such due to the existence of *self* steps. As the generated schema mapping is to be customized by human operators, it is desirable that the paths look "clean". A normalization processing can be applied to the paths resulting from the above processing to produce "clean" paths. As an example, a meta-level path expression such as `xspxshp:complexType/xspxshp:sequence/xspxshp:element` may become `./.fr:depth` after being converted to a schema-level path expression and can be simplified to `depth`.

The algorithm for deriving schema level *xsm:source* attributes of local target objects from a meta- level *xsm:source* attribute is summarized as a three-step process as shown in Figure 7.12.

For each *s*
    `msc` = evaluate context-dependent expressions in `ms`;
    `ss` = convert each meta-level path step in `msc` to a schema-level path step with information from *s*;
    `ss` = path-simplify `ss`;

Figure 7.12 Algorithm for deriving schema-level *xsm:source* attribute of a local target object from a meta-level *xsm:source* attribute.

(`ms` stands for the meta-level *xsm:attribute* being processed;
`s` stands for a matched node from the source schema tree;
`ss` stands for the resulting schema-level *xsm:attribute*.)

7.4.5 Other Mapping Annotations

Up to now, the focus has been on deriving schema-level *xsm:source* attributes. There are other mapping annotations to be generated. Their generation is straightforward as explained below.

The top-level *xsm:sourceSchema* element is filled with a source schema location obtained from the user.

The *xsm:root* element is specified by an meta-level *xsm:instanceRoot* element.

The common schema-level mapping file import specification, i.e. schema-level *xsm:import* common to all schema mappings of a meta-schema mapping between a pair of meta schemas, is specified using the introduced meta-level *xsm:instanceMappingImport* element.

As the schema-mapping processor and meta-schema mapping processor is the same program, an *xsm:mapping* element is introduced into meta-schema mappings to indicate the mapping level. In particular, the presence of attribute *level* = *"xsm:meta"* in a mapping indicates it is a meta-schema mapping instead of a schema mapping.

The mapping elements/attributes, namely *xsm:method, xsm:variable xsm:param* and *xsm:with-param* are not considered in the current implementation of meta-schema mapping processing as they are mainly used in customizing schema mappings. It may be desirable that a meta-schema mapping can take parameters to allow more convenient customization of meta-schema mappings. In such cases, those mapping objects can be helpful. This has not been explored in this study.

7.5 Case Study of Generating Schema Mappings

With the method developed above, the schema mapping processor introduced in the last chapter has been enhanced to process meta-schema mapping. That is, it now translates a meta-schema mapping to a schema mapping generator instead of a schema translator. The meta-schema mapping from XSHP meta-schema to XMIF meta-schema, i.e. `xshp-to-xmif4_1.xsm.xsd`, is presented in Appendix E. The meta-schema mapping has then been processed by the (meta-) schema mapping processor to produce the XSHP-to-XMIF schema mapping generator, `xshp-to-xmif.xsm.xslt`, which is presented in Appendix F. The generated XSHP-to-XMIF schema mapping generator has been tested with a number of test cases. One of them is presented here for demonstration.

```
<?xml version="1.0" encoding="UTF-8"?>
<xspxshp:schema xmlns="http://www.w3.org/2001/XMLSchema"
xmlns:xspxshp="http://www.xggt.org/xspxshp"
xmlns:xshp="http://www.xggt.org/xshp"
targetNamespace="http://www.xggt.org/example"
elementFormDefault="qualified"
attributeFormDefault="unqualified"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.xggt.org/xspxshp xshp4_1.xsd">

 <xspxshp:import namespace=http://www.xggt.org/xshp
 schemaLocation="xshpTypes3.xsd"/>

   <xspxshp:element name="Lake"
substitutionGroup="xshp:_Feature">
       <xspxshp:complexType>
         <xspxshp:sequence>
             <xspxshp:sequence>
               <xspxshp:element name="name" type="string"/>
                   <xspxshp:element name="area" type="decimal"/>
                   <xspxshp:element name="surf_elev"
type="decimal"/>
                   <xspxshp:element name="depth" type="decimal"/>
               </xspxshp:sequence>
             <xspxshp:element name="Shape"
             type="xshp:PolygonPropertyType"/>
         </xspxshp:sequence>
       </xspxshp:complexType>
   </xspxshp:element>
```

Figure 7.13 A sample XSPXSHP schema `xshpLake.xsd.xml`

The input is an XSPXSHP schema, `xshpLake.xsd.xml`, as shown in Figure 7.13. It has been processed by the XSHP-to-XMIF schema mapping generator and has resulted in the schema mapping shown in Figure 7.14, `xmifLake.xsd.xml`, an XSPXMIF schema annotated with mapping information. The resulting schema mapping can then either be customized by a human operator if wanted, or directly processed by the schema mapping processor to produce a data instance translator. The generated data instance translator can then be used to translate XSHP data, which conform to the WXS schema equivalent to `xshpLake.xsd.xml`, to XMIF data, which conform to the WXS schema equivalent to `xmifLake.xsd.xml`.

The complete process of translating XSHP data to XMIF data using the method of (meta-) schema mapping is illustrated in Figure 7.15. Comparing Figure 7.15 with

Figure 4.8 of Chapter 4, it is found that Figure 7.15 can be seen as an example of Figure 4.8, if the latter is seen as a template.

```
<?xml version="1.0" encoding="UTF-8"?>
<xspxmif:schema xmlns="http://www.w3.org/2001/XMLSchema"
xmlns:_tmp_From="http://www.xggt.org/example"
xmlns:_tmp_To="http://www.xggt.org/xspxmif-
idxspxshp:schema86847336" xmlns:fn="http://www.w3.org/2005/xpath-
functions" xmlns:xdt="http://www.w3.org/2005/xpath-datatypes"
xmlns:xmif="http://www.xggt.org/xmif"
xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:xshp="http://www.xggt.org/xshp"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
xmlns:xsm="http://www.xggt.org/mapping"
xmlns:xspxmif="http://www.xggt.org/xspxmif"
xmlns:xspxshp="http://www.xggt.org/xspxshp"
targetNamespace="http://www.xggt.org/xspxmif-
idxspxshp:schema86847336" elementFormDefault="qualified"
attributeFormDefault="unqualified"
xsi:schemaLocation="http://www.xggt.org/xspxmif ../xmif/xmif4_1.x
sd">

 <xspxmif:import namespace=http://www.xggt.org/xmif
 schemaLocation="xmifTypes4.xsd"/>
 <xspxmif:import namespace=http://www.xggt.org/xmif
 schemaLocation="xmifMeta4.xsd"/>

 <xspxmif:annotation xmlns="">
  <xspxmif:appinfo>
   <xsm:import schemaLocation="xmifMeta4.xsm.xsd"/>
  </xspxmif:appinfo>
 </xspxmif:annotation>

 <xspxmif:element xsm:source="_tmp_From:Lake" name="Lake"
 substitutionGroup="xmif:_Feature">
  <xspxmif:complexType>
   <xspxmif:sequence>
       <xspxmif:sequence>
      <xspxmif:element xsm:source="_tmp_From:name" name="name"
      type="xs:string"/>
        <xspxmif:element xsm:source="_tmp_From:area" name="area"
        type="xs:decimal"/>
        <xspxmif:element xsm:source="_tmp_From:surf_elev"
        name="surf_elev" type="xs:decimal"/>
        <xspxmif:element xsm:source="_tmp_From:depth" name="depth"
        type="xs:decimal"/>
       </xspxmif:sequence>
       <xspxmif:element xsm:source="_tmp_From:Shape"
name="Geometry"
    type="xmif:GeometryPropertyType"/>
   </xspxmif:sequence>
  </xspxmif:complexType>
 </xspxmif:element>
```

Figure 7.14 Generated schema mapping `xmifLake.xsd.xml` from `xshpLake.xsd.xml`

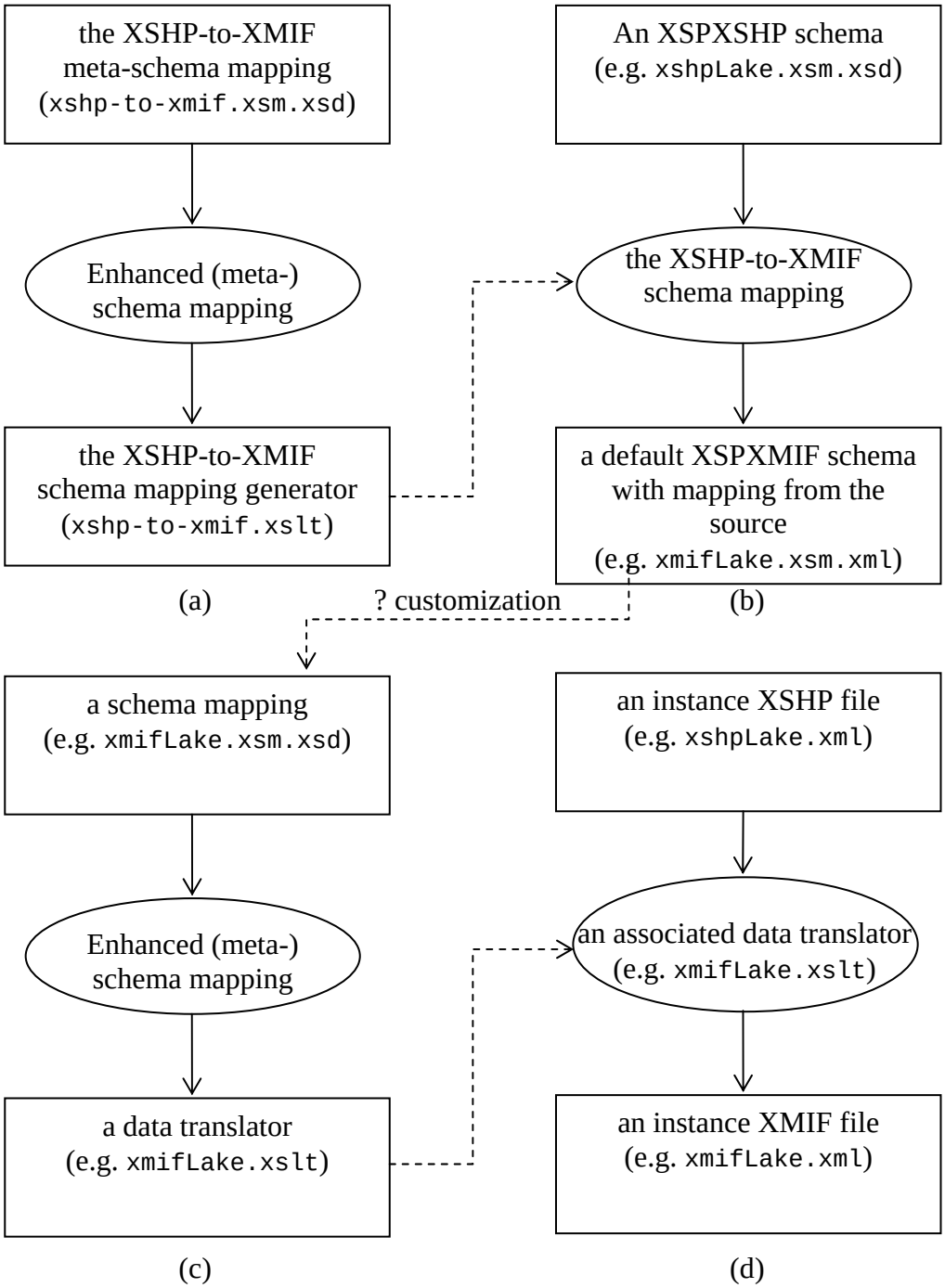


Figure 7.15 Translating XSHP data to XMIF data using the (meta-) schema

7.6 Comparison with Related Work

Data model or schema translation is of wide concern for database interoperability. Much work has been done on schema translation from one specific meta-model to another specific meta-model, for instance, from semantic data models to a specific

variant of relational models (Flynn and Laender 1985, Lyngbaek and Vianu 1987). There has also been much work on schema integration via a common canonical meta-model (Chomicki and Litwin 1993, Kelley 1995, Gingras et al. 1997).

Few works on general methods of schema translation can be found. Atzeni and Torlone (1995) shared the concern of this author. This author thinks that is due to the fact that it is widely believed that translation via a common canonical (meta-) model should be preferred to direct translation in order to save effort on building translators. That cannot be wrong. The problem is that there needs to be a means better than brute force programming to address the diversity in meta-models when no one common canonical model has been or can be agreed upon.

The work of Atzeni and Torlone (1993, 1995, 1996) is a rare serious study on a general method of schema translation and deserves a detailed introduction. They proposed a meta-meta-model approach for the management of multiple meta-models and model translation. (The terms involved have been converted according to the term convention used in this study. In their original terms, meta-model is used to refer to our meta-meta model; model is used to refer to our meta-model and schema is used to refer to our data model.) In their approach, a very small set of meta-meta-constructs are used in the meta-meta-model, which, depending on the set of meta-models to be dealt with, may include lexical type, abstract type, aggregation, generalization, function and grouping, etc. Meta-meta-constructs are then used as nodes and edges to compose structures, of which there are two kinds. One is called *pattern*, of which the edges are labelled with cardinality. A meta-model is defined by its allowed set of *patterns*, of which each corresponds to a meta-construct in the meta-model. The other kind of structure has nodes and edges labelled with names in

addition to the cardinality label of edges. Structures labelled with names are used to represent schemas. If a schema consists only of *patterns* allowed in a certain meta-model, then it is said to be a schema of that meta-model. By defining elementary transformations among meta-meta-constructs, the problem of schema translation can be resolved by composing a complex transformation using the elementary transformations according to the patterns allowed in a target meta-model. A set of elementary transformations is said to be complete with respect to a set of meta-models if it allows derivation of translations between any pair of meta-models in the set of concern. The search for a complete set of elementary transformations has to be done pragmatically. Given a complete set of elementary transformations, a method for generating translations between any pair of meta-models is proposed.

Compared with the approach adopted in this study, one superior characteristic of their approach is that schema translators can be generated automatically, while in the approach of this study the meta-schema mapping has to be specified before the schema translator can be generated. That characteristic reveals the fundamental difference between their approach and the approach adopted here. As was stated in Chapter 4, the approach of this study is based on the idea of syntax-direct translation. As a grammar only defines the syntactical semantics of a language, full semantics of sentences (i.e. meta-schemas in our discussion) are not present. Thus, in this approach, users are required to specify the semantic relationship between meta-constructs of different meta-models. In contrast, in the approach of Atzeni and Torlone (1995), the definition of meta-models is not a syntactical definition. Instead, it is a mathematical definition based on a very small set of meta-meta-constructs. That means the formal semantics of meta-models is fully captured. Therefore, it is possible to fully automate schema translation in their approach.

The framework proposed by Atzeni and Torlone (1993, 1995, 1996) is a promising one and worth further study. However, there are factors that prevent this author from exploring its application in this study. First of all, there are theoretical problems remaining in their framework. These include the setup of a meta-meta-model, the theoretical principle and practical guide on the search for a complete set of elementary transformations, and the principle of deriving an optimal translation from elementary transformations. In addition, their work is largely isolated from the problem of data instance translation. As this study aims to facilitate writing geodata translators, the problem of schema translation needs to be considered in a framework that addresses other matters as well, which includes the coordination of schema translators and data instance translators. Therefore, a more practical approach is adopted in this study.

That, however, does not mean this approach cannot be extended to support full automation of translation. Indeed, this author believes by introducing ontologies of modelling constructs at different layers, full semantics of data, schemas and meta-schemas can be defined for a community that share the ontologies and thus full automation of data translation is possible. However, the work of Atzeni and Torlone (1995) can be considered as a specific method of specifying a specific ontology for meta-modelling. In that sense, the approach of this study is more compatible with the current framework addressing semantics.

Another work related to the topic of this chapter is that of MDA (Model Driven Architecture). The problem of general schema translation is closely related to general methods of meta-schema specification. The latter is the basis of the former as there cannot be a general method of specifying schema translations without a general

method of specifying meta-schemas. As part of the MDA effort (OMG 2001c), a method of metadata specification is defined in MOF (meta-object facility) (OMG 2001b). In MOF, a meta-model is described using metadata classes, while a data model or schema is taken as data instances of metadata classes. All data, i.e. metadata and meta-metadata, can be encoded in a specially designed XML format called XMI (XML Metadata Interchange) (OMG 2003b). This is a general method of meta-model description, which, if applied recursively, can be used to introduce as many layers of modelling as needed. In current practice, it is found that four layers of modelling are enough from a conceptual point of view, while more layers are indeed used in implementation. As MDA is an ongoing effort, there has not been a mechanism to validate/manipulate models against/according to their associated meta-models except by operational programming. That is not a welcome property from the viewpoint of this study, as we want to enable high-level specification of translation. Although there have been proposals on specifying transformations of models defined using MOF (Gogolla, 2000, Peltier 2001, Sprinkle et al. 2003), none of them are suitable for the problem of data translation to the knowledge of this author. Again, the point is translators or transformers at one level of modelling are isolated from those at other layers. As pointed out by Dsouza and Gerber et al (Dsouza 2001, Gerber et al 2002), the key is to maintain links between models as well as translating models.

From the methodology point of view, the approach of this study is in essence the same as that of MDA. Regarding meta-schema specification, this study has avoided introducing a new meta-meta-formalism. Instead, the WXS is used recursively as the ultra formalism for specifying schemas at all modelling levels. This is similar to the approach taken in the MDA effort, in which a subset of UML (Unified Modelling

Language) is defined and called the MOF Model. The MOF Model is then used recursively to define itself and lower level models. Regarding model or schema translation, the approach adopted here is very similar to the proposal of Paige and Radjenovic (2003), which suggests using TXL for model transformation in MDA. A distinct characteristic of the approach of this study is that translators at different levels are not isolated but have been chained to automatically work together. In particular, a meta-level translator not only generates a target schema from a source schema but also generates the mapping between them. The mapping can then be used to generate translators at data instance level.

7.7 Tests on Real Datasets

Tests on the XML-based implementation of the proposed approach using real datasets have also been performed. The test program has been implemented using Visual C# .Net. The interface of the program is shown in the following figure.

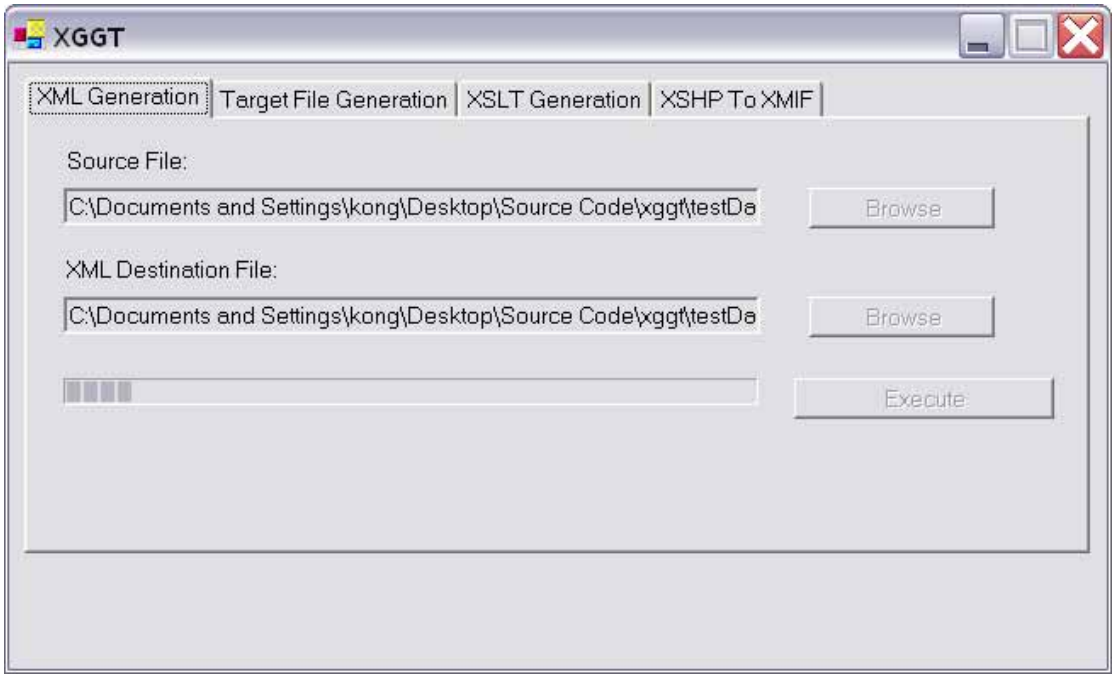


Figure 7.16 Interface of the test program

The program accepts SHAPE datasets and produces MIF datasets. For a SHPAE dataset, it first converts it to an XSHP file using the SHAPE-to-XSHP decoder mentioned in Chapter 5. The associated XSHP schema is also produced. From the XSHP schema, a default XMIF schema is generated together with the mappings between the two schemas by the generated schema mapping generator mentioned in Section 7.6. The generated schema mapping is then processed by the schema mapping processor to produce the data instance translator, which transforms the XSHP file into the intended XMIF file. Finally, the resulting XMIF file is converted into an MIF file by the XMIF-to-MIF encoder mentioned in Chapter 5.

A few real SHAPE datasets have been used to test the whole process. These files were of geometric types of point, polyline or polygon. Their sizes ranged from tens to hundreds of kilobytes. The intermediate XSHP or XMIF files went as large as over ten megabytes. The XSLT transformation performed took tens of seconds in average on a PC with 3.0GHz CPU and 1.0G RAM. The resulting MIF datasets have been imported into the MapInfo software to view and check the results. All of the tests were successful and therefore demonstrated that the approach and its XML-based implementation in whole works well.

7.8 Summary and Discussion

In this chapter, the meta-schema mapping method for schema translation is proposed. In the method, a meta-schema is specified as a WXS schema that validates a subset of WXS schemas that corresponds to all valid schemas of the specific meta-model. With two meta-schemas, one as the source and the other as the target, a meta-schema mapping can be defined using the schema-mapping method proposed in the last chapter. The meta-schema mapping can then be processed by the (meta-) schema

mapping processor to derive a schema translator from the source to the target. As has been shown, schema translation is not enough for the purpose of this study. A method of turning a schema translator to a schema-mapping generator was thus devised. Given a meta-schema mapping, the (meta-) schema mapping processor is enhanced to generate a schema mapping generator instead of a schema translator. A schema mapping generator is able to append mapping annotations to a target schema and thus turn it into a schema mapping. The key problem in the process is to generate *xsm:source* attributes when needed and derive their values, i.e. relative source paths with respect to their immediate upper level mapping. Algorithms for deriving *xsm:source* values have been devised and implemented.

As can be seen, translation at both data instance level and data schema level are addressed with the same method of SDT. Moreover, translators at each of the two levels are derived from mappings defined at the immediate upper level. The essence of SDT is to have syntax drive the translation process and have semantics specified as annotations to syntax.

Thanks to the adoption of XML syntax at all levels of modelling, including the data instance level, schema level and meta-schema level, translators at different levels are based on the same data manipulation method, i.e. XSLT. This has greatly facilitated the implementation of the (meta-) schema mapping processor.

The essence of this methodology to address the diversity of data encoding and modelling can be seen as a combination of using a common canonical model at each level while having the diversity defined at its immediate upper level. In this way, the method enjoys both the benefits of translation via a common canonical model and those of direct translation through the general methodology of SDT.

Chapter 8

Conclusions and Recommendations

This study originated from the need for better geodata exchange mechanisms and geodata translation technologies. In particular, it arose out of the observation of two barriers to geodata translation, the difficulty in developing geodata translators and the lack of mechanisms for performing semantic reconciliation. It thus aimed to develop techniques to automate geodata translator generation and introduce a mechanism for reconciling semantic differences in the process of geodata translation.

This study started with an investigation of issues involved in geodata translation. Then, a review of current geodata translation techniques was performed and their limitations pointed out. Inspired by the idea of MDA, this study proposed a model driven framework to tackle the two barriers identified. Finally, XML-based techniques were developed to support the implementation of the proposed framework.

8.1 Conclusions

The main contribution of this study is the proposed model-driven framework for geodata exchange and the techniques developed to support putting the framework into practice. In particular, the following conclusions can be drawn from this study:

The proposed framework for geodata exchange enables, by design, automatic generation of geodata translators and reconciliation of semantic heterogeneity

through declarative specification of custom transformations. It consists of the 4-layer model of data representation, the 3-layer model of geodata translators and the model-driven method of generating data instance translators. The 4-layer model of data representation allows it to accommodate various geodata formats. The 3-layer model of geodata translators and the model-driven method enables the generation of schema translators and data instance translators from high-level meta-schema mappings and schema mappings, respectively.

Although XML technology provides a basis for implementing the proposed framework, two limitations of current XML techniques have been identified with respect to the implementation. First, there is not a method for defining 4-layer XML formats, which also makes high-level specification of data model transformations not viable. Secondly, XSLT is not an ideal schema transformation language and the gap between XSLT and XML Schema needs to be tackled. Techniques have thus been developed to fill these blanks.

A method for developing 4-layer XML-based geodata formats has been proposed and demonstrated in this study. In the method, XML syntax is used as the common syntax of encoding data instance; WXS is used as the common schema definition language; a meta-schema is specified as a WXS schema that validates a subset of WXS schemas that correspond to all valid schemas of the specific meta-model.

A schema mapping specification method for XML document transformation has been proposed. It is based on WXS. Schema mapping is specified by annotating a target

schema with assertions of correspondence from a source schema. It enables a high level specification of geodata transformation without getting involved in data accessing details. Imperative programming is thus not needed for common data restructuring tasks. Although the proposed schema mapping method has not been developed to fully support WXS, its implemented subset shows that the idea of generating an XSLT stylesheet from schema mapping is beneficial. Algorithms for translating schema mapping specifications into XSLT stylesheets have been developed and implemented. The presented examples show that the schema mapping specification method is expressive and declarative.

With two meta-schemas, one as the source and the other as the target, a meta-schema mapping can be defined using the proposed schema-mapping method. A method of generating schema mapping generators from meta-schema mappings has been devised. A schema mapping generator is able to append mapping annotations to a target schema and thus turn it into a schema mapping. The key problem in the process is the generation of *xsm:source* attributes when needed and the derivation of their values, i.e. relative source paths with respect to their immediate upper level mapping. Algorithms for deriving *xsm:source* values have been devised and implemented.

A series of experiments were carried out following the proposed methodology and using the techniques that were developed. 4-layer XML counterparts of SHAPE and MIF formats were specified. A meta-schema mapping from XSHP to XMIF was

defined. It was converted to an XSHP-to-XMIF schema mapping generator using the mapping processor developed in this study. The generated schema mapping generator was used to derive XMIF schemas from XSHP schemas together with the associated schema mappings. Derived schema mappings, as well as specified schema mappings, were used to derive XSHP-to-XMIF data instance translators. The generated data instance translator together with the developed SHAPE-XSHP and XMIF-MIF encoders/decoders were used to translate SHAPE data into MIF data. The experiments demonstrated that the design goals of the framework had been achieved and the developed techniques worked well.

8.2 Limitations of this Study

Two key techniques have been developed in this study. They are methods for specifying (meta-) schema mappings and chaining meta-schema mappings to schema mappings. The method for specifying (meta-) schema mappings developed has not been designed to fully support WXS. As shown in this study, it is very beneficial to allow high-level declarative specification of data transformation. Further study on fully supporting WXS is desirable. In addition, a graphical user interface for specifying (meta-) schema mappings is also valuable for practical use. The developed method of chaining meta-schema mappings to schema mappings is interesting. It is based on the fact that WXS has been used to define a subset of itself. Only limited testing of the method has been performed in this study. It is possible that there are unrevealed issues. More work is needed to fully develop the method. This study has

not addressed the efficiency of XML-based geodata translation. Further study on optimizing XML document transformation is needed when the datasets involved are large. Generally, XML database technology should be helpful in dealing with large datasets.

8.3 Recommendations for Further Studies

In addition to improving this study from the above-mentioned aspects, further study on integrating the methodology and technology developed in this study with the technology of ontology is considered beneficial. In this study, it is assumed that human operators process data semantics. That process includes analysing the schemas, metadata, the associated documents, and so on to clarify the semantic relationships and converting the resulting semantic relationships into schema transformation specifications. It is desirable that this process can be automated. The ontology technology developed in the AI field has the potential to be of great help in the automation of the process.

As we know, current semantic data models only formalize the general relationships between concepts, such as generalization, aggregation and so on. Neither the meaning of a specific concept nor a specific relationship between concepts can be understood by a database system. To database systems, a *student* and a *person* are just different objects. It does not know how different they are. Further, although the database can know that a *student* IS A *person* and a *child* IS A *person* too by supporting the object-oriented data model, it cannot know of the relationship

between a *child* and a *student* in the context of a certain application. This is due to the fact that in database systems there is no mechanism to capture the meaning of concepts. Many geodata exchange standardization efforts have recognized the need for data producers and data users to use common definitions of concepts. As a result, a catalogue of feature and attribute definitions is included as part of the standards (Cascio 1994, USGS 1997b, DGIWG 1997d). Such a dictionary of definitions is, of course, necessary and helpful, but at the same time not formal, meaning that the vocabulary contained in the dictionary can only be processed by human operators, not by computers. As can be imagined, formal representation of vocabulary is an urgent need to capture more semantics of data and provide the basis of semantic reasoning.

In philosophy, ontology is the study of the kinds of things that exist (Grossmann 1992), while in the field of artificial intelligence the term *ontology* has largely come to mean one of two related things (Chandrasekaran et al. 1999). In one sense, ontology is a representation vocabulary (Gruber 1993, Chandrasekaran et al. 1999), often specialized to some domain or subject matter. More precisely, it is not the vocabulary as such that qualifies what is called an ontology, but the conceptualization that the terms in the vocabulary are intended to capture. In its second sense, the term ontology sometimes is used to refer to a body of knowledge describing some domain, typically a commonsense knowledge domain, using a representation vocabulary (Chandrasekaran et al. 1999). In other words, the representation vocabulary provides a set of terms with which to describe the facts in

some domain, while the body of knowledge using that vocabulary is a collection of facts about a domain (Chandrasekaran et al. 1999).

Being an intelligent vocabulary, the importance of ontology cannot be overestimated as it is a mechanism that can make computers be more comprehending. For geodata sharing, ontology has the potential to bring two main benefits. First, by building and sharing ontology for a domain or community of geographic information, fundamental semantic heterogeneity can be avoided (Bishr et al. 1999). Secondly, ontology can be used to automate semantic reasoning (Hakimpour and Timpf 2003). Applying ontology to assist semantic reasoning can be distinguished into two categories, namely, within-ontology semantic reasoning and inter-ontology reasoning (Guarino 1998). When the source and target data schema are defined using terms from the same ontology, the semantic relationships between them can be precisely reasoned out (MacGregor et al. 1997). When they are from different ontologies, the semantic relationship between two ontologies needs first to be defined (Parent and Spaccapietra 1998). With semantic relationships between a source schema and a target schema automatically reasoned out, human operators can confirm such assertions. Successive automatic process may derive schema mapping specifications from semantic relationship assertions (Pottinger and Bernstein 2003, Bowers and Ludäscher 2004) and hence can further automate geodata translation.

References

Agrawal A, T Levendovszky, J Sprinkle, F Shi, G Karsai, 2002, Generative Programming via Graph Transformations in the Model-Driven Architecture, *Workshop on Generative Techniques in the Context of Model Driven Architecture*, OOPSLA , Nov. 5, 2002, Seattle, WA.

Akman V and M Surav, 1996, Steps toward formalizing context, *AI Magazine*, 17(3):55-- 72

Alfelor R M, 1995, GIS and integrated highway information system, in *Sharing Geographic Information* (eds. H.J. Onsrud and R. Rushton), Rutgers, 397-411

Altheide P, 1992a, An implementation strategy for SDTS encoding, *Cartography and Geographic Information System*, Vol. 19, No. 5, pp306-310

Altheide P, 1992b, Design of a spatial data transfer processor, *Cartography and Geographic Information System*, Vol. 19, No. 5, pp311-314

Altheide P S, 1996, Development of a toolkit for Spatial Data Transfer Standard applications, *GIS/LIS'96 Proceedings, Annual Conference and Exhibition*, pp173-180

Altova, 2005, Data Integration: Opportunities, challenges, and Altova MapForce™, an Altova white paper, 18p.

Andries M, G.Engels, A Habel, B Hoffmann, H-J Kreowski, S Kuske, D Pump, A Schürr, and G Taentzer, 1999, Graph transformation for specification and programming, *Science of Computer Programming*, 34(1):1–54, Apr. 1999.

Antoniou G and F van Harmelen, 2004, A semantic Web primer, MIT Press, 238 p

Arctur D, D Hair, G Timson, E P Martin and R Fegeas, 1998, Issues and prospects for the next generation of the spatial data transfer standard (SDTS), *Int. J. Geographic Information Science*, Vol. 12, No. 4, 403-425.

Atzeni P and R Torlone, 1993, A metamodel approach for the management of multiple models and the translation of schemes, *Information Systems*, 18(6):349--362.

Atzeni P and R Torlone, 1995, Schema Translation between Heterogeneous Data Models in a Lattice Framework, in *Sixth IFIP TC-.2 Working Conference on Doto Semantics*

(DS-6), Atlanta, pages 218-227.

Atzeni P, R Torlone, 1996, Management of Multiple Models in an Extensible Database Design Tool, in *EDBT'96, LNCS 105'7*, Springer-Verlag, pages 79-95

Bailey I, 1995, The EXPRESS-M Reference Manual, available at http://www.cimio.co.uk

Bamberger W J, 1995, Sharing geographic information among local government agencies in the San Diego Region, in *Sharing Geographic Information* (eds. H.J. Onsrud and R. Rushton), Rutgers, 119-137

Barry D K, 1996, The object database handbook: how to select, implement, and use object-oriented databases, Wiley, 340p

Batini C, S Ceri, S B Navathe, 1992, Conceptual database design: an entity-relationship approach, Benjamin Cummings Pub. Co.

ter Bekke J H, 1992, Semantic data modeling, Prentice Hall, 272p

Bellahsène Z, T Milo, M Rys, D Suciu, R Unland (eds.), 2004, Database and XML technologies, *Second International XML Database Symposium, XSym 2004*, Toronto, Canada, August 29-30, proceedings, Springer-Verlag.

Berild S and B Wenzel, 1995, A comparative study of ISO/CEN geographic information reference models and STEP, ISO TC 211 N171, 62 pages

Bertino E and L Martino, 1993, Object-oriented database systems: concepts and architectures, Addison-Wesley Pub. Co., 264p

Bédard Y, 1999, Principles of spatial database analysis and design, in *Geographical Information Systems* (P A Longley, M F Goodchild, D J Maguire and D W Rhind, eds.), second edition, John wiley & Sons Inc., pp413-424

Birbeck M, J Diamond, J Duckett, O G. Gudmundsson, P Kobak, E Lenz, S Livingstone, D Marcus, S Mohr, N Ozu, J Pinnock, K Visco, A Watt, K Williams, and Z Zaev, 2001, Professional XML (2nd edition), Wrox Press Ltd.

Bishr Y A, H Pundt, W Kuhn, and M Radwan, 1999, Probing the concept of information communities - a first step toward semantic interoperability, in *Interoperating Geographic Information Systems* (M. Goodchild, Max Egenhofer, R. Fegeas, and C. Kottman, *editors*), Kluwer Academic, 55–69

Blaha M and W Premerlani, 1998, Object-oriented modeling and design for database

applications, Prentice Hall

Bornstein N M, 2003, .NET and XML, O'Reilly

Bowers S and B Ludäscher, 2004, An Ontology-Driven Framework for Data Transformation in Scientific Workflows, in E. Rahm (Ed.): *DILS 2004*, *LNBI 2994*, Springer-Verlag Berlin Heidelberg, pp1–16

Bukhres O A and A K Elmagarmid (editors), 1996, Object-oriented multidatabase systems, Englewood Cliffs, N.J.: Prentice Hall, 712p

Burrough P A and I Masser, 1998, European geographic information infrastructures: opportunities and pitfalls, London; Bristol, PA: Taylor & Francis

Burrough P A and R A McDonnell, 1998, Principles of geographical information systems, Oxford University Press

Carey M J, N M Mattos, and A K Nori, 1997, Object-relational database systems (tutorial): principles, products and challenges, *Proceedings of the 1997 ACM SIGMOD international conference on Management of data*

Cascio J, 1994, A spatial features Register: toward standardization of spatial features, *Cartography and Geographic Information Systems*, Vol. 21 No.3, pp155-158.

Cattell R G G and D K Barry (*eds.*), 1997, *The object database standard: ODMG 2.0*, Morgan Kaufmann Publishers, Inc.

Cattell R G G, D K Barry, et al. (Editor), 2000, The Object Data Standard: ODMG 3.0, Morgan Kaufmann, 300p

CEN TC 278, 1995, Geographic Data Files (first draft)

Chamberlin D, Fankhauser P, D Florescu, M Marchiori, J Robie, 2001, XML Query Use Cases, http://www.w3.org/TR/xquery-use-cases

Chandrasekaran B, J R Josephson, and V R Benjamins, 1999, What are ontologies and why do we need them? *IEEE Intelligent systems*, 1999, p20-26

Chaudhri A B, A Rashid and R Zicari (editors), 2003, XML data management: native XML and XML-enabled database systems, Addison-Wesley

Chomicki J and W Litwin, 1993, Declarative definition of object-oriented multidatabase mappings, in *Distributed Object Management* (M. T. Ozsu, U. Dayal, and P. Valduriez, *eds*.), Morgan-Kaufmann, Los Altos, Calif

Clark J and S DeRose (editors), 1999, XML Path Language (Version 1.0), W3C Recommendation, http://www.w3.org/TR/1999/REC-xpath-19991116

Colomb R M, 1997, Impact of semantic heterogeneity on federating databases, *COMPUTER JOURNAL* 40: (5) 235-244

Cooke D F, 1995, Sharing street centerline spatial databases, in *Sharing Geographic Information* (H.J. Onsrud and R. Rushton *eds*.), Rutgers, pp 363-376.

Craig W J, 1995, Why we can't share data: institutional inertia, in *Sharing Grographic Information* (eds. H.J. Onsrud and R. Rushton), Rutgers, pp107-118.

Date C J, 1990, An introduction to database systems (5th ed), Addison-Wesley

Coleman D J and J D McLaughlin, 1992, Standards for Spatial Information Interchange: A Management Perspective, *Canadian Institute for Surveying and Mapping Journal*, Vol. 46, No. 2, pp. 131-141.

Denno P (ed.), 1999, Product data representation and exchange - EXPRESS-X Language Reference Manual, (ISO TC184/SC4/WG11 N088)

Department of National Defense Canada (on behalf of DGIWG), 1999, Liaison contribution from DGIWG: The DIGEST Data Model in Terms of TC211

Devogele T, C Parent and S Spaccapietra, 1998, On spatial database integration, *IJGIS* vol.12, no.4, p335-352.

DGIWG (Digital Geographic Information Working Group), 1997, DIGEST (Digital Geographic Information Exchange Standard)
Edition 2.0, http://www.digest.org

DGIWG (Digital Geographic Information Working Group), 1997a, The Digital Geographic Information Exchange Standard (DIGEST): Part 1 - GENERAL DESCRIPTION, Edition 2.0

DGIWG (Digital Geographic Information Working Group), 1997b, The Digital Geographic Information Exchange Standard (DIGEST): Part 2 - THEORETICAL MODEL, EXCHANGE STRUCTURE AND ENCAPSULATION SPECIFICATIONS, Edition 2.0

DGIWG (Digital Geographic Information Working Group), 1997d, The Digital Geographic Information Exchange Standard (DIGEST): Part 4 - Feature and Attribute Coding Catalogue (FACC), Edition 2.0

DGIWG (Digital Geographic Information Working Group), 1999, The DIGEST data model in terms of TC211.

Dsouza D, 2001, Model-driven architecture and integration - opportunities and challenges, http://www.kinetium.com/catalysis-org/publications/papers/2001-mda-reqs-desmond-6.pdf.

DuCharme B, 2005, Automating Stylesheet Creation, http://www.xml.com/lpt/a/2005/09/07/autogenerating-xslt-stylesheets.html, last visited on Oct. 2005

Elmagarmid A, M Rusinkiewicz and A Sheth (eds.), 1999, Management of heterogeneous and autonomous database systems, Morgan Kaufmann, 413p

Elmasri R and S B Navathe, 2000, Fundamentals of database systems (3rd ed.), Addison-Wesley

ESRI, 1998, ESRI Shape file Technical Description --- An ESRI White Paper

Evans J and J Ferreira, 1995, Sharing spatial information in an imperfect world: interaction between technical and organizational issues, in *Sharing Grographic Information* (eds. H.J. Onsrud and R. Rushton), Rutgers, 363-376.

Fallside D C (ed.), 2001, XML Schema Part 0: Primer, W3C, 2 May 2001, http://www.w3.org/TR/2004/REC-xmlschema-0-20041028/primer.html

Feldman S and E Curtis, 2003, GI data sharing for e-Government: Using GML to make the vision a reality, *the AGI conference at GeoSolutions 2003*

Flynn D J and A H F Laender, 1985, Mapping from a conceptual schema to a target internal schema, *The Computer Journal*, vol. 28, no. 5

Fowler J, 1995, STEP for Data Management, Exchange and Sharing, Julian Fowler, 221p.

Fowler M and Kendall Scott, 2000, UML distilled: a brief guide to the standard object modeling language, Addison Wesley, 185p

Frank, A., 2003, Ontology for spatio-temporal Databases, in Spatiotemporal Databases: The Chorochronos Approach, M.e.a. Koubarakis, Editor. 2003, Springer, 2520: pp. 9-77.

Fry J P, D C P Smith, R W Taylor, R L Frank, V Y Lum, J A Behymer, and B Shneiderman, 1977, Stored-data description and data translation: A model and language, *Inform. Syst*. 2, 3 (1977), 95-160.

Gahegan M, 1996, Specifying the transformation within and between geographic data models, *Transactions in GIS*, Vol. 1, No. 2, 137-154.

GAI, 2004, GAI Meeting Minutes, Department of Interior, http://www.opengeospatial.org/press/?page=pressrelease&prid=111

GDBC (Geographic Data BC, Ministry of Environment, Lands and Parks, Province of British Columbia, Canada), 1994, SAIF Toolkit API Programmer's Reference Manual, http://srmwww.gov.bc.ca/bmgs/stk/index.html

GDBC (Geographic Data BC, Ministry of Environment, Lands and Parks, Province of British Columbia, Canada), 1995, SAIF (Spatial Archive and Interchange Format: Formal Definition Release 3.2), http://www.elp.gov.bc.ca/gdbc

Gerber A, M Lawley, K Raymond, J Steel, and A Wood, 2002, Transformation: The Missing Link of MDA, in *Proc. Graph Transformation - First International Conference* (A. Corradini, H. Ehrig, H.-J. Kreowski, and G. Rozenberg, editors), *ICGT 2002*

Gingras F, L V S Lakshmanan, I N Subramnnian, D Papoulis, and N Shiri 1997, Languages for multi-database interoperability, in *SIGMOD 1997, Proceedings ACMSIGMOD International Conference on Management of Data* (J. Peckham, *Ed.*), Tucson, Az., May 13–15, pp. 536–538.

Goh C H, S E Madnick , M D Siegel, 1994, Context interchange: overcoming the challenges of large-scale interoperable database systems in a dynamic environment, *Proceedings of the third international conference on Information and knowledge management*, p.337-346, November 29-December 02, 1994, Gaithersburg, Maryland, United States

Goh C H, S Bressan, S Madnick, M Siegel, 1999, Context interchange: New features and formalisms for the intelligent integration of information, *ACM TRANSACTIONS ON INFORMATION SYSTEMS* 17: (3) 270-293 JUL 1999

Gogolla M, 2000, Graph transformations on the UML metamodel, in *ICALP Workshop on Graph Transformations and Visual Modeling Techniques* (*Eds.* J. D. P. Rolim, A. Z. Broder, A. Corradini, R. Gorrieri, R. Heckel, J. Hromkovic, U. Vaccaro, J. B. Wells), Carleton Scientific, Waterloo, Ontario, Canada, 2000, pp. 359-371.

Goldfarb C F and P Prescod, 2002, The XML handbook (4th ed.), Prentice Hall

Goodchild M F and P A Longley, 1999, The future of GIS and spatial analysis, in *Geographical Information Systems* (P A Longley, M F Goodchild, D J Maguire and D W Rhind, eds.), second edition, John wiley & Sons Inc., pp567-580

Grossmann R, 1992, The existence of the world: an introduction to ontology, Routledge, London, 139p.

Gray P M D, Kulkarni K G and N W Paton, 1992, Object-oriented databases: a semantic data model approach, Prentice Hall, 237p.

Gruber T R, 1993, Towards principle for the design of ontology used for knowledge sharing, in *Formal Ontology in Conceptual Analysis and Knowledge Representation* (Guarino N and R Poli, *editors*), International Workshop on Ontology, Kluwer Academic.

Guarino N, 1998, Formal ontology and information systems, in *Formal Ontology in Information Systems* (Nicola Guarino, *editor*), Proceedings of FOIS'98, pages 3–17, Trento, Italy, June 1998. IOS Press, Amsterdam.

Gupta A, 1989, Integration of information systems: bridging heterogeneous databases, New York: Institute of Electrical and Electronics Engineers, 334p

Halevy A Y, Z G Ives, D Suciu, I Tatarinov, 2005, Schema mediation for large-scale semantic data sharing, *VLDB Journal* 14: 68–83

Harold E R, 1999, XML™ Bible, IDG Books Worldwide Inc.

Harold E R and W S Means, 2002, XML in a Nutshell (2nd Edition), O'Reilly, 634 Pages

Hohl P (ed.), 1998, GIS data conversion: strategies, techniques, and management, Santa Fe, N.M.: OnWord Press, 411p

Holzner S, 2003, XPath: Navigating XML with XPath 1.0 and 2.0 Kick Start, Sams Publishing, 600 Pages

Hurson A R, M W Bright, S Pakzad (eds.), 1994, Multidatabase systems: an advanced solution for global information sharing, Los Alamitos, Calif.: IEEE Computer Society, 387p

IBM, 2004, DB2 Spatial extender and geodetic extender user's guide and reference

IHO S-57, 1996, IHO (INTERNATIONAL HYDROGRAPHIC ORGANIZATION)

TRANSFER STANDARD for DIGITAL HYDROGRAPHIC DATA Edition 3.0 Special Publication No. 57, Published by the International Hydrographic Bureau, MONACO

ISO 10303-11, 1994, Product data representation and exchange - Description methods: The EXPRESS language reference manual

ISO 10303-22, 1994, Product data representation and exchange - Implementation methods: Standard data access interface

ISO 15046-1.2, 1998, CD 15046-1.2 Geographic information - Part 1: Reference model (ISO/TC 211 N 623)

ISO 15046-18, 1999, CD 15046-18 Geographic information - Part 18: Encoding (ISO/TC 211 N709)

ISO15046-9, CD 15046-9 Geographic information - Part 9: Rules for application schema (ISO/TC 211 N 631)

ISO/IEC 8211, 1994, Information Technology - Specification for a Data Descriptive File for Information Interchange

ISO 15046, Geographic Information/Geoinformatics, http://www.statkart.no/isotc211/

Jagadish H V, S Al-Khalifa, A Chapman1, L V S Lakshmanan, A Nierman1, S Paparizos, J M Patel, D Srivastava, NWiwatwattana, Y Q Wu, C Yu, 2002, TIMBER: A native XML database, *The VLDB Journal* 11: 274–291

Katz H (editor), 2003, XQuery from the Experts: A Guide to the W3C XML Query Language, Addison Wesley, 512 Pages

Kawaguchi K, 2001, W3C XML Schema Made Simple, published on *XML.com* http://www.xml.com/pub/a/2001/06/06/schemasimple.html

Kay M, 2001, XSLT Programmer's Reference, Wrox Press

Keller S E, J A Perkins, T F Payton, S P Mardinly, 1984, Tree transformation techniques and experiences, *ACM SIGPLAN Notices*, Volume 19, Issue 6 (June 1984), Proceedings of the SIGPLAN '84 symposium on compiler construction, Pages: 190 - 201

Kelley W, Gala S K, Kim W, T C Reyes and B Graham, 1995, Schema architecture of the UniSQL/M multidatabase system, in *Modern Database Systems*,

Addison-Wesley, Reading, Mass

Kim W and J Seo, 1991, Classifying schematic and data heterogeneity in multidatabase systems, IEEE Computer, 24:12, pp. 12-18.

Kleppe A, J Warmer and W Bast, 2003, "MDA Explained: the Practice and Promise of Model-Driven Architecture", Addison-Wesley

Krupnikov K A and H Thompson, 2001, Data Binding Using W3C XML Schema Annotations, *Proceedings of XML Conference & Exposition 2001*, December 9-14, Orlando, FL

Kuhn, W., 2002, Modeling the Semantics of Geographic Categories through Conceptual Integration. in Geographic Information Science - Second International Conference, GIScience, 2002, Boulder, CO, USA, September 2002. 2002: Springer, Lecture Notes in Computer Science 2478: pp. 108-118.

Laurini R, 1998, Spatial multi-database topological continuity and indexing: a step toward seamless GIS data interoperability, *Int. J. Geographic Information Science*, Vol. 12, No. 4, pp373-402

Lazar R A, 1992, SDTS Support Software: The FIPS 123 Function Library, *Cartography and Geographic Information Systems*, Vol. 19, No. 5, pp. 303-305

Lee D, M Mani, F Chiu and W W Chu, 2001, Nesting-based Relational-to-XML Schema Translation, in *Int'l Workshop on the Web and Databases (WebDB)*, Santa Barbara, CA, May 2001.

Lee D and W W Chu, 2000, Comparative Analysis of Six XML Schema Languages, *ACM SIGMOD Record*, 29(3):76-87

Lee J L, S E Madnick, M D Siegel, 1996, Conceptualizing semantic interoperability: A perspective from the knowledge level, *INTERNATIONAL JOURNAL OF COOPERATIVE INFORMATION SYSTEMS* 5: (4) 367-393 DEC 1996

Lee Y C, 1990, An object-oriented environment for GIS data exchange, *GIS for the 1990s*

Lee Y C and Coleman D J, 1990, A Framework for Evaluating Interchange Standards, *Canadian Institute for Surveying and Mapping Journal*, Vol. 44, No. 4, pp. 391-402.

Lee Y C and Z Xu, 1999, Design of a geodata translation system, *International Archives of Photogrammetry and Remote Sensing*, Vol32. Part 4W12, Joint ISPRS Commission workshop on Dynamic and Multi-Dimensional GIS, Oct. 1999, Beijing,

China, pp181-188.

Loffredo D, 1998, Efficient Database Implementation of EXPRESS Information Models, PhD Thesis, Rensselaer Polytechnic Institute, Troy, New York

Lyngbaek P and V Vianu, 1987, Mapping a semantic database model to the relational model, *Proc. of ACM SIGMOD*

MacGregor R M, H Chalupsky and E R Melz, 1997, PowerLoom Manual, http://www.isi.edu/isd/LOOM/Power-Loom/documentation/manual.pdf

Malhotra A and M Maloney (eds.), 1999, XML Schema Requirements, W3C Note

Mangano S, 2002, XSLT Cookbook, O'Reilly, 670 Pages

MapInfo Corporation, 1999, The MapInfo Interchange File (MIF) Format Specification

Loomis M E S, 1995, Object databases: the essentials, Addison-Wesley, 230p

Loomis M E S and A B Chaudhri., 1998, Object databases in practice, Prentice Hall PTR, 312p.

Masser I, 1999, All shapes and sizes: the first generation of national spatial data infrastructures, *Int. J. GIS*, Vol. 13, No.1, pp67-84

McLaughlin B, 2001, Java & XML (2nd Edition), O'Reilly, 528 pages

Mellor S J, K Scott, A Uhl and D Weise, 2004, MDA distilled: principles of model-driven architecture, Boston: Addison-Wesley

Microsoft, 2001, SQL Server 2000 XML Overview, URL: http://www.microsoft.com/technet/prodtechnol/sql/2000/evaluate/xmlsql.mspx

Miline P, S Milton and J Smith, 1993, Geographic object-oriented databases – a case study, *Int. J. Geographic Information Systems*, Vol.7, No.1, pp39-55

Miller J and J Mukerji (eds.), 2003, MDA Guide Version 1.0.1, OMG

Moellering H (ed.), 1991, Spatial database transfer standards: current international status, published on behalf of the International Cartographic Association by Elsevier Applied Science

Moellering H, 1994, Continuing Research Needs Resulting from the SDTS,

Cartography and Geographic Information Systems, Vol. 21, No. 3 (July, 1994), pp. 180-187

Moellering H and R Hogan (eds.), 1997, Spatial Database Transfer Standards 2: Characteristics for Assessing Standards and Full Descriptions of the National and International Standards in the world, published on behalf of the International Cartographic Association by Elsevier Applied Science.

Mulberry Technologies, Inc., 2000, Data Modeling and XML Vocabulary Development, http://www.mulberrytech.com/papers/xmlvocab.pdf

Navathe S B and J P Fry, 1976, Restructuring for Large Databases: Three Levels of Abstraction, ACM Transactiona on Database Systems. Vol. 1, No. 2, June 1976, Pages 138-158.

Navathe S B and A Savasere, 1996, A schema integration facility using object-oriented data model, in *Object-oriented multi-database systems* (*edited by Omran A. Bukhres, Ahmed K. Elmagarmid)*, Englewood Cliffs, N.J.: Prentice Hall, pp105-127

National Research Council (United States of America), 2001, National spatial data infrastructure partnership programs: rethinking the focus, National Academy Press, Washington, D.C.

OASIS Technical Committee, 2002, Relax NG, available at: http://www.oasis-open.org/committees/relax-ng/

Obasanjo D, 2002, W3C XML Schema Design Patterns: Avoiding Complexity, http://www.xml.com/pub/a/2002/11/20/schemas.html, last visited on November 20, 2005

Obasanjo D, 2003, XML Schema Design Patterns: Is Complex Type Derivation Unnecessary? http://www.xml.com/pub/a/2003/10/29/derivation.html

OGC, 1998, The OpenGIS Guide (Third Edition, Eds. Buehler, K. and McKee, L.), http://www.ogis.org

OGC, 1999, The OpenGIS Abstract Specification Model, http://www.ogis.org

OGC, 2002, OpenGIS® Geography Markup Language (GML) Implementation Specification, version 2.1.2 (Simon Cox, Adrian Cuthbert, Ron Lake, Richard Martell *editors*), http://www.opengeospatial.org/docs/02-069.pdf

OMG, 2001a, Interchange Metamodel in XML, OMG Document: formal/01-02-15,

Feb. 2001.

OMG, 2001b, Meta Object Facility (MOF) v1.3.1, OMG Document: formal/01-11-02, Nov. 2001.

OMG, 2001c, Model Driven Architecture – A Technical Perspective, OMG Document: ormsc/01-
07-01, July 2001.

OMG, 2001d, Unified Modeling Language v1.4, OMG Document: formal/01-09-67, Sept. 2001.

OMG, 2003a, Common Warehouse Metamodel (CWM), http://doc.omg.org/formal/03-03-02

OMG, 2003b, XML Metadata Interchange (XMI), v2.0 specification (formal/03-05-01)

Oracle, 1999a, All Your Data: The Oracle Extensibility Architecture, an Oracle Technical White Paper

Oracle, 1999b, Bringing Objects to the Mainstream, an Oracle White Paper

Oracle, 2001, Simple Strategies for Complex Data: Oracle9i Object-Relational Technology, an Oracle Technical White Paper

Oracle 2005a, Oracle Spatial 10g—Technical White Paper (PDF)

Oracle, 2005b, XML technology center, URL: http://www.oracle.com/oramag/mar02/index.html?xsu.html

Paige R and A Radjenovic, 2003, Towards Model Transformation with TXL, in *Proc. Metamodelling for MDA Workshop 2003*, York, UK, November 2003, available at http://www-users.cs.york.ac.uk/~paige/pubs.html

Pankowski T, 2003a, Transformation of XML data using an unranked tree transducer, *E-Commerce and Web Technologies, 4th International Conference, EC-Web 2003*, Lecture Notes in Computer Science Vol. 2738, p259--269

Pankowski T, 2003b, Specifying transformations for XML data, in *Proceedings of the Pre-Conference Workshop of VLDB 2003, Emerging Database Research in East Europe*, 86-90

Parent C and S Spaccapietra, 1998, Issues and approaches of database integration,

Communications of the ACM, 41(5):166–178

Pascoe R and J Penny, 1990, Construction of interfaces for the exchange of geographic data, *Int. J. Geographic Information Systems*, Vol.4, No.2, pp147-156

Pascoe R and J Penny, 1995, Constructing interfaces between (and within) geographical information systems, *Int. J. Geographic Information Science*, Vol.9, No.3, pp275-291

Peltier M, J Bézivin, and G Guillaume, 2001, MTRANS: A general framework based on XSLT for model transformations, in *WTUML'01, Proceedings of the Workshop on Transformations in UML*, Genova, Italy, April 2001

Poole J D, 2001, Model-Driven Architecture: Vision, Standards, and Emerging Technologies, *ECOOP 2001 Workshop on Metamodeling and Adaptive Object Models*

Pottinger R and P A Bernstein, 2003, Merging models based on given correspondences, In *Proceedings of the 29th International Conference on Very Large Data Bases (VLDB)*, Morgan Kaufmann, pages 826–837

Provost W, 2002, Working with a Metaschema, Published on XML.com http://www.xml.com/pub/a/2002/10/02/metaschema.html

Reddy M P and A Gupta, 1995, Context interchange – a lattice based approach, *Knowledge-based Systems*, 8: (1) 5-13

Revault N, 2003, Model transformation based on production rules, electronic notes in *Theoretical Computer Science* 72 No. 4 (2003), URL: http://www.elsevier.nl/locate/entcs/volume72.html 14 pages

Rhind D W, 1999, National and international geospatial data polices, in Geographical Information Systems (P A Longley, M F Goodchild, D J Maguire and D W Rhind, eds.), second edition, John wiley & Sons Inc.

Robertsson E, 2003, An Introduction to Schematron, Published on XML.com http://www.xml.com/pub/a/2003/11/12/schematron.html

Safe Software, 2005a, FME Product Brochure, http://www.safe.com

Safe Software, 2005b, Semantic Translation, http://www.safe.com

Safe Software, 2005c, Semantic Data Translation Using FME, http://www.safe.com

Safe Software, 2005d, Formats Supported: Product Comparison Chart, http://www.safe.com

Salgé F, 1999 National and international data standards, in Geographical Information Systems (P A Longley, M F Goodchild, D J Maguire and D W Rhind, eds.), second edition, John wiley & Sons Inc.

Sanders G L, 1995, Data modeling, Boyd & Fraser Pub. Co.

Schenck D and P Wilson, 1994, Information modeling: the EXPRESS way, Oxford University Press, 388p

Scholl M and A Voisard, 1991, Object-oriented database systems for geographic applications: an experiment with O2, *Geographic database management systems, workshop proceedings*, Capri, Italy, May 1991, Springer-Verlag.

Shepherd I D H, 1999, Information integration and GIS, in Geographical Information Systems (P A Longley, M F Goodchild, D J Maguire and D W Rhind, eds.), second edition, John wiley & Sons Inc.

Sheth A and J Larson, 1990, Federated database systems for managing distributed, heterogeneous, and autonomous databases, *ACM Comput Surv* 22(3): 183–236

Shu N C, B C Housel, R W Taylor, S P Ghosh, and V Y Lum, 1977, EXPRESS: A Data Extraction, Processing, and REStructuring System, *ACM Tranrurctions on Database System*, Vol. 2, No. 2. June 1977, Pages 134-174.

Skonnard A and M Gudgin, 2001, Essential XML Quick Reference, Addison-Wesley

Smith J M, 1992, SGML and related standards, Ellis Horwood, 151p

Smith N S and D W Rhind 1999, Characteristics and sources of framework data, in Geographical Information Systems (P A Longley, M F Goodchild, D J Maguire and D W Rhind, eds.), second edition, John wiley & Sons Inc., pp655-666.

Sperling J, 1995, Development and maintenance of the TIGER database: experiences in spatial data sharing at the U.S. Bureau of the Census, in *Sharing Grographic Information* (eds. H.J. Onsrud and R. Rushton), Rutgers, 377-396.

Sprinkle J, A Agrawal, T Levendovszky, F Shi, and G Karsai, 2003, Domain Model Translation Using Graph Transformations, in *10th IEEE International Conference and Workshop on the Engineering of Computer-Based Systems (ECBS'03)*, p.159

Stigberg D, 1994, An SDTS implementation for GRASS, *Cartography and*

Geographic Information Systems, Vol. 21 No. 3, pp162-171

Sun Microsystems, Java 2 Enterprise Edition (J2EE), http://java.sun.com/j2ee/

Tang X and F W Tompa, 2001, Specifying transformations for structured documents, in *Proceedings of the 4th International Workshop on the Web and Databases* (Mecca G and J Simeon *eds*.), WebDB, 67–72

Thompson H S, D Beech, M Maloney, and N Mendelsohn (editors), 2001, XML Schema Part 1: Structures, W3C Recommendation, http://www.w3.org/TR/2001/REC-xmlschema-1-20010502/

Ullman J D and J Widom, 1997, A first course in database systems,
Upper Saddle River, N.J.: Prentice Hall, 470p

Unknown (Cover Pages), 2005, W3C Workshop to Address Improved Interoperability of Schema-Aware Software http://xml.coverpages.org/ni2005-05-24-a.html

USGS, 1997a, Spatial Data Transfer Standard (SDTS) - Part 1, Logical Specifications, American National Standards Institute, Inc.

USGS, 1997b, Spatial Data Transfer Standard (SDTS) - Part 2, Spatial Features, American National Standards Institute, Inc.

USGS, 1997c, Spatial Data Transfer Standard (SDTS) - Part 3, ISO 8211 Encoding, American National Standards Institute, Inc.

USGS, 1997d, Spatial Data Transfer Standard (SDTS) - Part 4 Topological Vector Profile, American National Standards Institute, Inc.

Uitermark H, A Vogels and P Oosterom, 1999, Semantic and geometric aspects of integrating road networks, *Interoperating geographic information systems------Proceedings of Second International Conference INTEROP'99*, Zurich, Switzerland, 1999

van der Vlist E, 2001, Comparing XML Schema Languages, Published on XML.com http://www.xml.com/pub/a/2001/12/12/schemacompare.html

van der Vlist E, 2002, XML Schema O'Reilly & Associates, Inc.

W3C, 2002, XML Schema, W3C Recommendation, available at: http://www.w3.org/XML/Schema

W3C DOM Working Group, 2002, Document object model, available at: http://www.w3.org/DOM/

Wayne L D, Randolph A. McBride, Michael Carpenter, Matteson W. Hiland, Christine Todd, and S. Jeffress Williams, 1994, Integrating Standards for Cataloging Geospatial Data, *Proceedings of GIS/LIS'94*, 25-27 Oct., 1994, Phoenix, Arizona.

Weibel R, 1998, Computational Perspectives on Map Generalization, *GeoInformatica* 2:4, 307-314.

Williams M G, 1992, Conversion of a U.S. Geological Survey DLG-3 Data Set to the SDTS Topological Vector Profile, *Cartography and Geographic Information Systems*, Vol. 19, No. 5 (December, 1992), pp. 315-320.

Williams R J, 1993, Digital Geographic Data Exchange Standards and Products: Description, Comparison and Opinions, *Cartography* Vol. 22 No.1.

Wirth N, 1996, Compiler construction, Harlow: Addison-Wesley, 176p.

Wong C, 2003, Overview of DB2's XML Capabilities: An introduction to SQL/XML functions in DB2 UDB and the DB2 XML Extender, URL: http://www-128.ibm.com/developerworks/db2/library/techarticle/dm-0311wong/

Wyke R A, S Rehman and Brad Leupen, 2001, XML Programming (Core Reference), Microsoft Press

XML Path Language (XPath) 2.0, 2002, W3C Working Draft, http://www.w3.org/TR/xpath20

XSL Transformations (XSLT) v1.0, 1999, W3C Recommendation, http://www.w3.org/TR/xslt

XSL Transformations (XSLT) 2.0, 2002, W3C Working Draft, http://www.w3.org/TR/xslt20

XQuery 1.0: An XML Query Language, 2002, W3C Working Draft, http://www.w3.org/TR/xquery

XQuery 1.0 and XPath 2.0 Data Model, 2002, W3C Working Draft, http://www.w3.org/TR/query-datamodel

Xu Z, Y C Lee and Y Q Chen, 2000, Schema transformation for semantic geodata translation, *International Archives of Photogrammetry and Remote Sensing*, Vol. XXXIII, Part B6. Amsterdam, July 2000, pp195-202

Z Xu and Y C Lee, 2002a, Networking GIS: issues, models and a review, *Proceedings of the 95th Annual Geomatics Conference of Canadian Institution of Geomatics*, Ottawa, Canada, Jul. 2002

Z Xu and YC Lee, 2002b, Semantic heterogeneity of geodata, *International Archives of Photogrammetry and Remote Sensing*, Vol. 34, Part 4

Z Xu and Y C Lee, 2002c, Network enabling GIS, GIM International, Vol. 16(11)

Yergeau F, T Bray, J Paoli, C M Sperberg-McQueen, E Maler (eds.), 2004, Extensible Markup Language (XML) 1.0 (Third Edition), W3C Recommendation, http://www.w3.org/TR/2004/REC-xml-20040204/

Appendix A

Comparing 2-layer and 3-layer XML-based data formats

Part A. 2-layer XML-encoding of relational data

Library-2layer.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<database name="library"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="tab2Layer.xsd">
 <table name="book">
  <pKey name="ISBN">
   <fieldName>ISBN</fieldName>
  </pKey>
  <record>
   <field name="ISBN" type="nvchar(10)">1233435566</field>
   <field name="title" type="nvchar(255)">Life is to love</field>
   <field name="press" type="nvchar(255)">apress</field>
  </record>
  <record>
   <field name="ISBN" type="nvchar(10)">1233435567</field>
   <field name="title" type="nvchar(255)">Life is to create</field>
   <field name="press" type="nvchar(255)">apress</field>
  </record>
 </table>
 <table name="author">
  <pKey name="id">
   <fieldName>id</fieldName>
  </pKey>
  <record>
   <field name="id" type="integer">12345</field>
   <field name="first_name" type="nvchar(50)">Frank A.</field>
   <field name="last_name" type="nvchar(50)">Mike</field>
   <field name="email" type="nvchar(255)">frank.a.mike@mail.com</field>
  </record>
  <record>
   <field name="id" type="integer">12346</field>
   <field name="first_name" type="nvchar(50)">Frank B.</field>
   <field name="last_name" type="nvchar(50)">Mike</field>
   <field name="email" type="nvchar(255)">frank.b.mike@mail.com</field>
  </record>
 </table>
 <table name="book_author">
  <pKey>
   <fieldName>ISBN</fieldName>
   <fieldName>author_id</fieldName>
  </pKey>
  <fKey>
   <tableName>book</tableName>
   <pKeyName>ISBN</pKeyName>
   <fieldName>ISBN</fieldName>
  </fKey>
  <fKey>
   <tableName>author</tableName>
   <pKeyName>id</pKeyName>
   <fieldName>author_id</fieldName>
  </fKey>
  <record>
   <field name="ISBN" type="nvchar(10)">1233435566</field>
   <field name="author_id" type="integer">12345</field>
```

```
  </record>
  <record>
   <field name="ISBN" type="nvchar(10)">1233435567</field>
   <field name="author_id" type="integer">12345</field>
  </record>
  <record>
   <field name="ISBN" type="nvchar(10)">1233435567</field>
   <field name="author_id" type="integer">12346</field>
  </record>
 </table>
</database>
```

Tab2layer.xsd

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
elementFormDefault="qualified">
 <xs:element name="database">
  <xs:complexType>
   <xs:sequence>
    <xs:element ref="table" minOccurs="0" maxOccurs="unbounded"/>
   </xs:sequence>
   <xs:attribute name="name" type="xs:token" use="required"/>
  </xs:complexType>
 </xs:element>
 <xs:element name="table">
  <xs:complexType>
   <xs:sequence>
    <xs:element ref="pKey" minOccurs="0"/>
    <xs:element ref="fKey" minOccurs="0" maxOccurs="unbounded"/>
    <xs:element ref="record" minOccurs="0" maxOccurs="unbounded"/>
   </xs:sequence>
   <xs:attribute name="name" type="xs:token" use="required"/>
  </xs:complexType>
 </xs:element>
 <xs:element name="pKey">
  <xs:complexType>
   <xs:sequence>
    <xs:element name="fieldName" type="xs:token" maxOccurs="unbounded"/>
   </xs:sequence>
   <xs:attribute name="name" type="xs:token"/>
  </xs:complexType>
 </xs:element>
 <xs:element name="fKey">
  <xs:complexType>
   <xs:sequence>
    <xs:element name="tableName" type="xs:token"/>
    <xs:element name="pKeyName" type="xs:token"/>
    <xs:element name="fieldName" type="xs:token" maxOccurs="unbounded"/>
   </xs:sequence>
  </xs:complexType>
 </xs:element>
 <xs:element name="record">
  <xs:complexType>
   <xs:sequence>
    <xs:element ref="field" maxOccurs="unbounded"/>
   </xs:sequence>
  </xs:complexType>
 </xs:element>
 <xs:element name="field">
  <xs:complexType mixed="true">
   <xs:attribute name="name" type="xs:token" use="required"/>
   <xs:attribute name="type" type="xs:token" use="required"/>
  </xs:complexType>
 </xs:element>
</xs:schema>
```

Part B. 3-layer XML-encoding of relational data

library.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<Library xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.xggt.org/table/example-library library.xsd"
xmlns="http://www.xggt.org/table/example-library">
 <books>
    <book>
            <ISBN>1233435566</ISBN>
            <title>Life is to love</title>
            <press>apress</press>
        </book>
        <book>
            <ISBN>1233435567</ISBN>
            <title>Life is to create</title>
            <press>apress</press>
        </book>
    </books>
    <authors>
        <author>
            <id>12345</id>
            <first_name>Frank A.</first_name>
            <last_name>Mike</last_name>
            <email>frank.a.mike@mail.com</email>
        </author>
        <author>
            <id>12346</id>
            <first_name>Frank B.</first_name>
            <last_name>Mike</last_name>
            <email>frank.b.mike@mail.com</email>
        </author>
    </authors>
    <book_author_relation>
        <book_author>
            <ISBN>1233435566</ISBN>
            <author_id>12345</author_id>
        </book_author>
        <book_author>
            <ISBN>1233435567</ISBN>
            <author_id>12345</author_id>
        </book_author>
        <book_author>
            <ISBN>1233435567</ISBN>
            <author_id>12346</author_id>
        </book_author>
    </book_author_relation>
</Library>
```

library.xsd

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns="http://www.xggt.org/table/example-library"
targetNamespace="http://www.xggt.org/table/example-library"
elementFormDefault="qualified" attributeFormDefault="unqualified">
 <xs:element name="Library">
  <xs:complexType>
   <xs:sequence>
    <xs:element name="books">
     <xs:complexType>
      <xs:sequence>
```

```
        <xs:element name="book" maxOccurs="unbounded">
         <xs:complexType>
          <xs:sequence>
           <xs:element name="ISBN" type="xs:string"/>
           <xs:element name="title" type="xs:string"/>
           <xs:element name="press" type="xs:string"/>
          </xs:sequence>
         </xs:complexType>
        </xs:element>
      </xs:sequence>
     </xs:complexType>
    </xs:element>
    <xs:element name="authors">
     <xs:complexType>
      <xs:sequence>
       <xs:element name="author" maxOccurs="unbounded">
        <xs:complexType>
         <xs:sequence>
          <xs:element name="id" type="xs:integer"/>
          <xs:element name="first_name" type="xs:token"/>
          <xs:element name="last_name" type="xs:token"/>
          <xs:element name="email" type="xs:string"/>
         </xs:sequence>
        </xs:complexType>
       </xs:element>
      </xs:sequence>
     </xs:complexType>
    </xs:element>
    <xs:element name="book_author_relation">
     <xs:complexType>
      <xs:sequence>
       <xs:element name="book_author" maxOccurs="unbounded">
        <xs:complexType>
         <xs:sequence>
          <xs:element name="ISBN" type="xs:string"/>
          <xs:element name="author_id" type="xs:integer"/>
         </xs:sequence>
        </xs:complexType>
       </xs:element>
      </xs:sequence>
     </xs:complexType>
    </xs:element>
   </xs:sequence>
  </xs:complexType>
  <xs:key name="pkAuthor">
   <xs:selector xpath="authors/author"/>
   <xs:field xpath="id"/>
  </xs:key>
  <xs:key name="pkBook">
   <xs:selector xpath="books/book"/>
   <xs:field xpath="ISBN"/>
  </xs:key>
  <xs:keyref name="isbnRef" refer="pkBook">
   <xs:selector xpath="book_author_relation/book_author"/>
   <xs:field xpath="ISBN"/>
  </xs:keyref>
  <xs:keyref name="authorIdRef" refer="pkAuthor">
   <xs:selector xpath="book_author_relation/book_author"/>
   <xs:field xpath="author_id"/>
  </xs:keyref>
 </xs:element>
</xs:schema>
```

Appendix B

XSHP Spatial Schema

xshpTypes3.xsd

```
<?xml version="1.0" encoding="UTF-8"?>
<schema xmlns="http://www.w3.org/2001/XMLSchema"
xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:xshp="http://www.xggt.org/xshp"
targetNamespace="http://www.xggt.org/xshp" elementFormDefault="qualified"
attributeFormDefault="unqualified">
 <!-- the file structure -->
 <element name="XshpFile">
  <complexType>
   <sequence>
    <element name="shpVersion" type="xs:string"/>
    <element name="xshpVersion" type="xs:string"/>
    <element ref="xshp:_Feature" maxOccurs="unbounded"/>
   </sequence>
  </complexType>
 </element>
 <element name="_Feature" abstract="true"/>
 <!-- the following are geometry properties and their associated types -->
 <element name="_geometryProperty" type="xshp:GeometryPropertyType"
abstract="true"/>
 <element name="pointProperty" type="xshp:PointPropertyType"
substitutionGroup="xshp:_geometryProperty"/>
 <element name="polylineProperty" type="xshp:PolylinePropertyType"
substitutionGroup="xshp:_geometryProperty"/>
 <element name="polygonProperty" type="xshp:PolygonPropertyType"
substitutionGroup="xshp:_geometryProperty"/>
 <element name="pointMProperty" type="xshp:PointMPropertyType"
substitutionGroup="xshp:_geometryProperty"/>
 <element name="multiPointProperty" type="xshp:MultiPointPropertyType"
substitutionGroup="xshp:_geometryProperty"/>
 <complexType name="GeometryPropertyType" abstract="true">
  <sequence minOccurs="0">
   <element ref="xshp:_Geometry"/>
  </sequence>
 </complexType>
 <complexType name="PointPropertyType">
  <complexContent>
   <restriction base="xshp:GeometryPropertyType">
    <sequence>
     <element ref="xshp:Point"/>
    </sequence>
   </restriction>
  </complexContent>
 </complexType>
 <complexType name="PolylinePropertyType">
  <complexContent>
   <restriction base="xshp:GeometryPropertyType">
    <sequence>
     <element ref="xshp:Polyline"/>
    </sequence>
   </restriction>
  </complexContent>
 </complexType>
 <complexType name="PolygonPropertyType">
  <complexContent>
   <restriction base="xshp:GeometryPropertyType">
    <sequence>
```

```
    <element ref="xshp:Polygon"/>
   </sequence>
  </restriction>
 </complexContent>
</complexType>
<complexType name="PointMPropertyType">
 <complexContent>
  <restriction base="xshp:GeometryPropertyType">
   <sequence>
    <element ref="xshp:PointM"/>
   </sequence>
  </restriction>
 </complexContent>
</complexType>
<complexType name="MultiPointPropertyType">
 <complexContent>
  <restriction base="xshp:GeometryPropertyType">
   <sequence>
    <element ref="xshp:MultiPoint"/>
   </sequence>
  </restriction>
 </complexContent>
</complexType>
<!-- the following are geometric objects -->
<element name="_Geometry" type="xshp:AbstractGeometryType" abstract="true"/>
<element name="Point" type="xshp:PointType"
substitutionGroup="xshp:_Geometry"/>
<element name="MultiPoint" type="xshp:MultiPointType"
substitutionGroup="xshp:_Geometry"/>
<element name="Polyline" type="xshp:PolylineType"
substitutionGroup="xshp:_Geometry"/>
<element name="Polygon" type="xshp:PolygonType"
substitutionGroup="xshp:_Geometry"/>
<element name="PointM" type="xshp:PointMType"
substitutionGroup="xshp:_Geometry"/>
<element name="MultiPointM" type="xshp:MultiPointMType"
substitutionGroup="xshp:_Geometry"/>
<!-- the following are supporting geometric objects -->
<element name="Coordinate2d" type="xshp:Coordinate2dType"/>
<element name="Box" type="xshp:BoxType"/>
<element name="LineString" type="xshp:LineStringType"/>
<element name="OuterRing" type="xshp:OuterRingType"/>
<element name="InnerRing" type="xshp:InnerRingType"/>
<element name="vertexes">
 <complexType>
  <sequence>
   <element ref="xshp:Coordinate2d" minOccurs="2" maxOccurs="unbounded"/>
  </sequence>
 </complexType>
</element>
<element name="componentPoints">
 <complexType>
  <sequence>
   <element ref="xshp:Coordinate2d" minOccurs="2" maxOccurs="unbounded"/>
  </sequence>
 </complexType>
</element>
<!-- the following are supporting geometric properties -->
<element name="boundingBox">
 <complexType>
  <sequence>
   <element ref="xshp:Box"/>
  </sequence>
 </complexType>
</element>
<!-- the following are supporting geometric data type definitions -->
<complexType name="Coordinate2dType">
 <sequence>
```

```
  <element name="x" type="double"/>
  <element name="y" type="double"/>
 </sequence>
</complexType>
<complexType name="BoxType">
 <sequence>
  <element name="corners">
   <complexType>
    <sequence>
     <element ref="xshp:Coordinate2d" minOccurs="2" maxOccurs="2"/>
    </sequence>
   </complexType>
  </element>
 </sequence>
</complexType>
<complexType name="LineStringType">
 <sequence>
  <element ref="xshp:vertexes"/>
 </sequence>
</complexType>
<complexType name="LineRingType" abstract="true">
 <complexContent>
  <restriction base="xshp:LineStringType">
   <sequence>
    <element ref="xshp:vertexes"/>
   </sequence>
  </restriction>
 </complexContent>
</complexType>
<complexType name="PolygonBoundaryRingType">
 <complexContent>
  <restriction base="xshp:LineRingType">
   <sequence>
    <element ref="xshp:vertexes"/>
   </sequence>
  </restriction>
 </complexContent>
</complexType>
<complexType name="OuterRingType">
 <annotation>
  <documentation>
  the vertex of a outer ring should be ordered clockwise
  </documentation>
 </annotation>
 <complexContent>
  <restriction base="xshp:PolygonBoundaryRingType">
   <sequence>
    <element ref="xshp:vertexes"/>
   </sequence>
  </restriction>
 </complexContent>
</complexType>
<complexType name="InnerRingType">
 <annotation>
  <documentation>
  vertex of a inner ring should be ordered counterclockwise
  </documentation>
 </annotation>
 <complexContent>
  <restriction base="xshp:PolygonBoundaryRingType">
   <sequence>
    <element ref="xshp:vertexes"/>
   </sequence>
  </restriction>
 </complexContent>
</complexType>
<!-- the following are geometric data type definitions -->
<complexType name="AbstractGeometryType" abstract="true">
```

```
 <complexContent>
  <restriction base="anyType">
   <attribute name="gid" type="ID" use="optional"/>
  </restriction>
 </complexContent>
</complexType>
<complexType name="PointType">
 <complexContent>
  <extension base="xshp:AbstractGeometryType">
   <sequence>
    <element name="coord">
     <complexType>
      <sequence>
       <element ref="xshp:Coordinate2d"/>
      </sequence>
     </complexType>
    </element>
   </sequence>
  </extension>
 </complexContent>
</complexType>
<complexType name="MultiPointType">
 <complexContent>
  <extension base="xshp:AbstractGeometryType">
   <sequence>
    <element ref="xshp:boundingBox"/>
    <element ref="xshp:componentPoints"/>
   </sequence>
  </extension>
 </complexContent>
</complexType>
<complexType name="PolylineType">
 <complexContent>
  <extension base="xshp:AbstractGeometryType">
   <sequence>
    <element ref="xshp:boundingBox"/>
    <element name="lineStrings">
     <complexType>
      <sequence>
       <element ref="xshp:LineString" maxOccurs="unbounded"/>
      </sequence>
     </complexType>
    </element>
   </sequence>
  </extension>
 </complexContent>
</complexType>
<complexType name="PolygonType">
 <complexContent>
  <extension base="xshp:AbstractGeometryType">
   <sequence>
    <element ref="xshp:boundingBox"/>
    <element name="boundaryRings">
     <complexType>
      <sequence>
       <element ref="xshp:OuterRing"/>
       <choice minOccurs="0" maxOccurs="unbounded">
        <element ref="xshp:OuterRing"/>
        <element ref="xshp:InnerRing"/>
       </choice>
      </sequence>
     </complexType>
    </element>
   </sequence>
  </extension>
 </complexContent>
</complexType>
<complexType name="PointMType">
```

```
  <complexContent>
   <extension base="xshp:PointType">
    <sequence>
     <element name="measure" type="double"/>
    </sequence>
   </extension>
  </complexContent>
 </complexType>
 <complexType name="MultiPointMType">
  <complexContent>
   <extension base="xshp:AbstractGeometryType">
    <sequence>
     <element ref="xshp:boundingBox"/>
     <element name="mMin" type="double"/>
     <element name="mMax" type="double"/>
     <element name="pointsMeasured">
      <complexType>
       <sequence>
        <element ref="xshp:PointM" maxOccurs="unbounded"/>
       </sequence>
      </complexType>
     </element>
    </sequence>
   </extension>
  </complexContent>
 </complexType>
</schema>
```

Appendix C

XMIF Spatial Schema

xmifTypes4.xsm.xsd

```
<?xml version="1.0" encoding="UTF-8"?>
<schema xmlns:xmif="http://www.xggt.org/xmif"
xmlns:xshp="http://www.xggt.org/xshp"
xmlns="http://www.w3.org/2001/XMLSchema"
xmlns:xsm="http://www.xggt.org/mapping"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
targetNamespace="http://www.xggt.org/xmif" elementFormDefault="qualified"
attributeFormDefault="unqualified">
 <!-- the following are geometric properties -->
 <annotation>
  <appinfo>
   <xsm:source schemaLocation="../xshp/xshpTypes3.xsd"/>
   <xsl:import href="xmifFromXshpHelper.xslt"/>
  </appinfo>
 </annotation>
 <element name="geometryProperty" type="xmif:GeometryPropertyType"
xsm:source=""/>
 <complexType name="GeometryPropertyType" xsm:source="">
  <choice>
   <element ref="xmif:Point" xsm:source="xshp:Point | xshp:PointM |
xshp:Coordinate2d"/>
   <element ref="xmif:Line"/>
   <element ref="xmif:Polyline" xsm:source="xshp:Polyline"/>
   <element ref="xmif:Region" xsm:source="xshp:Polygon"/>
   <element ref="xmif:Ellipse"/>
   <element ref="xmif:EllipticArc"/>
   <element ref="xmif:Rectangle"/>
   <element ref="xmif:RoundedRectangle"/>
  </choice>
 </complexType>
 <!-- the following are geometry objects -->
 <element name="_Geometry" type="xmif:AbstractGeometryType" abstract="true"/>
 <element name="Point" type="xmif:PointType"
substitutionGroup="xmif:_Geometry" xsm:source="xshp:Point"/>
 <element name="_PointM" type="xmif:PointType"
substitutionGroup="xmif:_Geometry" xsm:target="Point"
xsm:source="xshp:PointM" xsm:method="xmif:Point_from_xshp_PointM"/>
 <element name="_Coordinate2d" type="xmif:PointType"
substitutionGroup="xmif:_Geometry" xsm:target="Point"
xsm:source="xshp:Coordinate2d"
xsm:method="xmif:Point_from_xshp_Coordinate2d"/>
 <element name="Line" type="xmif:LineType"
substitutionGroup="xmif:_Geometry"/>
 <element name="Polyline" type="xmif:PolylineType"
substitutionGroup="xmif:_Geometry" xsm:source="xshp:Polyline"/>
 <element name="Region" type="xmif:RegionType"
substitutionGroup="xmif:_Geometry" xsm:source="xshp:Polygon"/>
 <element name="Ellipse" type="xmif:EllipseType"
substitutionGroup="xmif:_Geometry"/>
 <element name="EllipticArc" type="xmif:EllipticArcType"
substitutionGroup="xmif:_Geometry"/>
 <element name="Rectangle" type="xmif:RectangleType"
substitutionGroup="xmif:_Geometry"/>
 <element name="RoundedRectangle" type="xmif:RoundedRectangleType"
substitutionGroup="xmif:_Geometry"/>
 <!-- the following are symbolic objects -->
 <element name="_Symbol" abstract="true"/>
```

```
 <element name="Pen" type="xmif:PenType" substitutionGroup="xmif:_Symbol"/>
 <element name="Brush" type="xmif:BrushType"
substitutionGroup="xmif:_Symbol"/>
 <element name="PointSymbolCustomBitmapFile"
type="xmif:PointSymbolCustomBitmapFileType"/>
 <element name="PointSymbolMapinfoV3" type="xmif:PointSymbolMapinfoV3Type"/>
 <element name="PointSymbolTrueTypeFont"
type="xmif:PointSymbolTrueTypeFontType"/>
 <element name="PointSymbol" substitutionGroup="xmif:_Symbol">
  <complexType>
   <choice>
    <element ref="xmif:PointSymbolCustomBitmapFile"/>
    <element ref="xmif:PointSymbolTrueTypeFont"/>
    <element ref="xmif:PointSymbolMapinfoV3"/>
   </choice>
  </complexType>
 </element>
 <!-- the following are symbolic properties -->
 <element name="_symbolProperty" abstract="true"/>
 <element name="pointSymbol" substitutionGroup="xmif:_symbolProperty">
  <complexType>
   <sequence>
    <element ref="xmif:PointSymbol"/>
   </sequence>
  </complexType>
 </element>
 <element name="pen" substitutionGroup="xmif:_symbolProperty">
  <complexType>
   <sequence>
    <element ref="xmif:Pen"/>
   </sequence>
  </complexType>
 </element>
 <element name="brush" substitutionGroup="xmif:_symbolProperty">
  <complexType>
   <sequence>
    <element ref="xmif:Brush"/>
   </sequence>
  </complexType>
 </element>
 <!-- the following are supporting types -->
 <complexType name="Coord2dType" xsm:source="">
  <sequence>
   <element name="x" type="double" xsm:source="xshp:x"/>
   <element name="y" type="double" xsm:source="xshp:y"/>
  </sequence>
 </complexType>
 <complexType name="LineStringType" xsm:source="">
  <sequence>
   <element ref="xmif:vertexes" xsm:source="xshp:vertexes"/>
  </sequence>
 </complexType>
 <complexType name="LinearRingType" xsm:source="">
  <complexContent>
   <restriction base="xmif:LineStringType">
    <sequence>
     <element ref="xmif:vertexes" xsm:source="xshp:vertexes"/>
    </sequence>
   </restriction>
  </complexContent>
 </complexType>
 <!-- the following are supporting properties -->
 <element name="position" xsm:source="">
  <complexType>
   <sequence>
    <element ref="xmif:Coord2d" xsm:source="xshp:Coordinate2d"/>
   </sequence>
  </complexType>
```

```
 </element>
 <element name="vertexes" xsm:source="xshp:vertexes">
  <complexType>
   <sequence>
    <element ref="xmif:Coord2d" minOccurs="2" maxOccurs="unbounded"
xsm:source="xshp:Coordinate2d"/>
   </sequence>
  </complexType>
 </element>
 <element name="lineStrings" xsm:source="">
  <complexType>
   <sequence>
    <element ref="xmif:LineString" maxOccurs="unbounded"
xsm:source="xshp:LineString"/>
   </sequence>
  </complexType>
 </element>
 <element name="boundaryRings" xsm:source="xshp:boundaryRings">
  <complexType>
   <sequence>
    <element ref="xmif:LinearRing" maxOccurs="unbounded"
xsm:source="xshp:OuterRing | xshp:InnerRing"/>
   </sequence>
  </complexType>
 </element>
 <element name="diagonalCorners">
  <complexType>
   <sequence>
    <element ref="xmif:Coord2d" minOccurs="2" maxOccurs="2"/>
   </sequence>
  </complexType>
 </element>
 <!-- the following are supporting objects -->
 <element name="Coord2d" type="xmif:Coord2dType"
xsm:source="xshp:Coordinate2d"/>
 <element name="LineString" type="xmif:LineStringType"
xsm:source="xshp:LineString"/>
 <element name="LinearRing" type="xmif:LinearRingType"
xsm:source="xshp:OuterRing | xshp:InnerRing"/>
 <!-- the follwoing are geometric data types -->
 <complexType name="AbstractGeometryType" abstract="true" xsm:source="">
  <attribute name="gid" type="string" use="optional" xsm:source="@gid"/>
 </complexType>
 <complexType name="PointType" xsm:source="">
  <complexContent>
   <extension base="xmif:AbstractGeometryType">
    <sequence>
     <element ref="xmif:position" xsm:source="xshp:coord"/>
     <element ref="xmif:pointSymbol" minOccurs="0"/>
    </sequence>
   </extension>
  </complexContent>
 </complexType>
 <complexType name="LineType">
  <complexContent>
   <extension base="xmif:AbstractGeometryType">
    <sequence>
     <element ref="xmif:position" minOccurs="2" maxOccurs="2"/>
     <element ref="xmif:pen" minOccurs="0"/>
    </sequence>
   </extension>
  </complexContent>
 </complexType>
 <complexType name="PolylineType" xsm:source="">
  <complexContent>
   <extension base="xmif:AbstractGeometryType">
    <sequence>
     <element ref="xmif:lineStrings" xsm:source="xshp:lineStrings"/>
```

```
    <element ref="xmif:pen" minOccurs="0"/>
    <element name="smooth" type="boolean" minOccurs="0" xsm:source="$smooth"/>
   </sequence>
  </extension>
 </complexContent>
</complexType>
<complexType name="RegionType" xsm:source="">
 <complexContent>
  <extension base="xmif:AbstractGeometryType">
   <sequence>
    <element ref="xmif:boundaryRings" xsm:source="xshp:boundaryRings"/>
    <element ref="xmif:pen" minOccurs="0"/>
    <element ref="xmif:brush" minOccurs="0"/>
   </sequence>
  </extension>
 </complexContent>
</complexType>
<complexType name="EllipticArcType">
 <complexContent>
  <extension base="xmif:AbstractGeometryType">
   <sequence>
    <element name="boundingRectangleOfEllipse">
     <complexType>
      <sequence>
       <element ref="xmif:Rectangle"/>
      </sequence>
     </complexType>
    </element>
    <element name="startingAngleCounterClockwiseFrom3oClock" type="double"/>
    <element name="endingAngleCounterClockwiseFrom3oClock" type="double"/>
    <element ref="xmif:pen" minOccurs="0"/>
   </sequence>
  </extension>
 </complexContent>
</complexType>
<complexType name="RectangleType">
 <complexContent>
  <extension base="xmif:AbstractGeometryType">
   <sequence>
    <element ref="xmif:diagonalCorners"/>
    <element name="pen" type="xmif:PenType" minOccurs="0"/>
    <element name="brush" type="xmif:BrushType" minOccurs="0"/>
   </sequence>
  </extension>
 </complexContent>
</complexType>
<complexType name="RoundedRectangleType">
 <complexContent>
  <extension base="xmif:AbstractGeometryType">
   <sequence>
    <element ref="xmif:diagonalCorners"/>
    <element name="degreeOfRoundingInCoordinateUnit" type="double"/>
    <element name="pen" type="xmif:PenType" minOccurs="0"/>
    <element name="brush" type="xmif:BrushType" minOccurs="0"/>
   </sequence>
  </extension>
 </complexContent>
</complexType>
<complexType name="EllipseType">
 <complexContent>
  <extension base="xmif:AbstractGeometryType">
   <sequence>
    <element ref="xmif:diagonalCorners"/>
    <element name="pen" type="xmif:PenType" minOccurs="0"/>
    <element name="brush" type="xmif:BrushType" minOccurs="0"/>
   </sequence>
  </extension>
 </complexContent>
```

```
</complexType>
<!-- the following are symbolic types -->
<complexType name="AbstractSymbolType" abstract="true"/>
<complexType name="PenType">
 <complexContent>
  <extension base="xmif:AbstractSymbolType">
   <sequence>
    <element name="width" type="short"/>
    <element name="pattern" type="short"/>
    <element name="color" type="int"/>
   </sequence>
  </extension>
 </complexContent>
</complexType>
<complexType name="BrushType">
 <complexContent>
  <extension base="xmif:AbstractSymbolType">
   <sequence>
    <element name="pattern" type="byte"/>
    <element name="foreColor" type="int"/>
    <element name="backColor" type="int" minOccurs="0"/>
   </sequence>
  </extension>
 </complexContent>
</complexType>
<complexType name="AbstractPointSymbolType">
 <complexContent>
  <extension base="xmif:AbstractSymbolType"/>
 </complexContent>
</complexType>
<complexType name="PointSymbolMapinfoV3Type">
 <complexContent>
  <extension base="xmif:AbstractPointSymbolType">
   <sequence>
    <element name="shape" type="int"/>
    <element name="color" type="int"/>
    <element name="size" type="byte"/>
   </sequence>
  </extension>
 </complexContent>
</complexType>
<complexType name="PointSymbolTrueTypeFontType">
 <complexContent>
  <extension base="xmif:AbstractPointSymbolType">
   <sequence>
    <element name="shape" type="int"/>
    <element name="color" type="int"/>
    <element name="size" type="byte"/>
    <element name="fontName" type="string"/>
    <element name="fontStyle" type="int"/>
    <element name="rotation" type="double"/>
   </sequence>
  </extension>
 </complexContent>
</complexType>
<complexType name="PointSymbolCustomBitmapFileType">
 <complexContent>
  <extension base="xmif:AbstractSymbolType">
   <sequence>
    <element name="fileName" type="string"/>
    <element name="color" type="int"/>
    <element name="size" type="byte"/>
    <element name="customStyle" type="byte"/>
   </sequence>
  </extension>
 </complexContent>
</complexType>
<complexType name="FontType">
```

```
  <sequence>
   <element name="fontName" type="string"/>
   <element name="style" type="int"/>
   <element name="size" type="byte"/>
   <element name="foreColor" type="int"/>
   <element name="backColor" type="int" minOccurs="0"/>
  </sequence>
 </complexType>
 <!-- the following are some simple types -->
 <simpleType name="nonspatialPropertyTypeEnumeration">
  <restriction base="string">
   <enumeration value="char"/>
   <enumeration value="integer"/>
   <enumeration value="short"/>
   <enumeration value="decimal"/>
   <enumeration value="float"/>
   <enumeration value="double"/>
   <enumeration value="date"/>
   <enumeration value="boolean"/>
  </restriction>
 </simpleType>
 <simpleType name="nonspatialPropertyNameType">
  <restriction base="string">
   <maxLength value="16"/>
  </restriction>
 </simpleType>
 <simpleType name="spatialPropertyNameType">
  <restriction base="string">
   <enumeration value="spatialProperty"/>
  </restriction>
 </simpleType>
 <simpleType name="featureNameType">
  <restriction base="string">
   <maxLength value="16"/>
  </restriction>
 </simpleType>
</schema>
```

xmifMeta.xsm.xsd

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns="http://www.xggt.org/xmif"
xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:xmif="http://www.xggt.org/xmif" xmlns:xshp="http://www.xggt.org/xshp"
xmlns:xsm="http://www.xggt.org/mapping"
targetNamespace="http://www.xggt.org/xmif" elementFormDefault="qualified"
attributeFormDefault="unqualified">
 <xs:include schemaLocation="../xmif/xmifTypes4.xsd"/>
 <!-- Xmif file object and its associated type -->
 <xs:annotation>
  <xs:appinfo>
   <xsm:import schemaLocation="xmifTypes4.xsm.xsd"/>
   <xsm:root name="xmif:XmifFile"/>
  </xs:appinfo>
 </xs:annotation>
 <xs:element name="XmifFile" xsm:source="xshp:XshpFile">
  <xs:complexType>
   <xs:sequence>
    <xs:element name="version" type="xs:string" minOccurs="0"/>
    <xs:group ref="xmif:metaInfo" minOccurs="0"/>
    <xs:element ref="_Feature" maxOccurs="unbounded" xsm:source=""/>
   </xs:sequence>
  </xs:complexType>
 </xs:element>
 <!-- feautre object -->
 <xs:element name="_Feature" abstract="true" xsm:source=""/>
```

```
 <!-- feature collection -->
 <!-- feature types -->
 <xs:group name="metaInfo">
  <xs:sequence>
   <xs:element ref="xmif:coordinateSystem" minOccurs="0"/>
   <xs:element ref="xmif:transform" minOccurs="0"/>
  </xs:sequence>
 </xs:group>
 <xs:element name="coordinateSystem" abstract="true"/>
 <xs:element name="coordNonEarth" substitutionGroup="coordinateSystem">
  <xs:complexType>
   <xs:sequence>
    <xs:element name="units" type="UnitType" default="meters"/>
    <xs:element name="bounds">
     <xs:complexType>
      <xs:sequence>
       <xs:element name="minx" type="xs:double"/>
       <xs:element name="miny" type="xs:double"/>
       <xs:element name="maxx" type="xs:double"/>
       <xs:element name="maxy" type="xs:double"/>
      </xs:sequence>
     </xs:complexType>
    </xs:element>
   </xs:sequence>
  </xs:complexType>
 </xs:element>
 <xs:element name="coordSysEarth" substitutionGroup="coordinateSystem">
  <xs:complexType>
   <xs:sequence>
    <xs:element name="projection" type="xs:integer"/>
    <xs:element name="datuam" type="xs:integer"/>
    <xs:element name="unit" type="UnitType"/>
   </xs:sequence>
  </xs:complexType>
 </xs:element>
 <xs:simpleType name="UnitType">
  <xs:restriction base="xs:string">
   <xs:enumeration value="centimeters"/>
   <xs:enumeration value="meters"/>
   <xs:enumeration value="feet"/>
   <xs:enumeration value="yards"/>
   <xs:enumeration value="inches"/>
  </xs:restriction>
 </xs:simpleType>
 <xs:element name="transform">
  <xs:complexType>
   <xs:sequence>
    <xs:element name="xMultiplier" type="xs:double"/>
    <xs:element name="yMultiplier" type="xs:double"/>
    <xs:element name="xDisplacement" type="xs:double"/>
    <xs:element name="yDisplacement" type="xs:double"/>
   </xs:sequence>
  </xs:complexType>
 </xs:element>
</xs:schema>
```

Appendix D

XSHP Meta-Schema

xshp4_1.xsd

```
<?xml version="1.0" encoding="UTF-8"?>
<schema xmlns="http://www.w3.org/2001/XMLSchema"
xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:xspxshp="http://www.xggt.org/xspxshp"
xmlns:xshp="http://www.xggt.org/xshp"
targetNamespace="http://www.xggt.org/xspxshp" elementFormDefault="qualified"
attributeFormDefault="unqualified">
 <annotation>
  <documentation>
 meta-schema defined in XML Schema.
  </documentation>
 </annotation>
 <element name="schema">
  <complexType>
   <sequence>
    <element ref="xspxshp:import"/>
    <element ref="xspxshp:annotation" minOccurs="0" maxOccurs="unbounded"/>
    <element name="element" type="xspxshp:FeatureClass"/>
   </sequence>
   <attribute name="attributeFormDefault" type="xspxshp:formEnum"
default="unqualified"/>
   <attribute name="elementFormDefault" type="xspxshp:formEnum"
default="unqualified"/>
   <attribute name="targetNamespace" type="anyURI"/>
   <attribute name="version" type="token"/>
   <anyAttribute namespace="##other" processContents="skip"/>
  </complexType>
 </element>
 <complexType name="FeatureClass">
  <sequence>
   <element ref="xspxshp:annotation" minOccurs="0" maxOccurs="unbounded"/>
   <element name="complexType">
    <complexType>
     <sequence>
      <element ref="xspxshp:annotation" minOccurs="0" maxOccurs="unbounded"/>
      <element name="sequence">
       <complexType>
        <sequence>
         <element ref="xspxshp:annotation" minOccurs="0"
maxOccurs="unbounded"/>
         <element name="sequence" type="xspxshp:FieldSequence"/>
         <element name="element" type="xspxshp:G_Field"/>
        </sequence>
       </complexType>
      </element>
     </sequence>
    </complexType>
   </element>
  </sequence>
  <attribute name="name" type="NCName"/>
  <attribute name="substitutionGroup" type="QName" fixed="xshp:_Feature"/>
 </complexType>
 <complexType name="FieldSequence">
  <sequence>
   <choice minOccurs="0" maxOccurs="unbounded">
    <element ref="xspxshp:annotation"/>
    <element name="element" type="xspxshp:Field"/>
```

```
   </choice>
  </sequence>
 </complexType>
 <complexType name="Field">
  <attribute name="name" type="NCName"/>
  <attribute name="type" type="xspxshp:SimpleTypeEnum"/>
  <anyAttribute namespace="##other" processContents="skip"/>
 </complexType>
 <complexType name="G_Field">
  <attribute name="name" type="NCName" fixed="Shape"/>
  <attribute name="type" type="xspxshp:ShapeTypeEnum"/>
  <anyAttribute namespace="##other" processContents="skip"/>
 </complexType>
 <element name="import">
  <complexType>
   <sequence minOccurs="0">
    <element ref="xspxshp:annotation"/>
   </sequence>
   <attribute name="id" type="ID"/>
   <attribute name="namespace" type="string"
fixed="http://www.xggt.org/xshp"/>
   <attribute name="schemaLocation" type="string" default="xshpTypes3.xsd"/>
  </complexType>
 </element>
 <simpleType name="formEnum">
  <restriction base="string">
   <enumeration value="qualified"/>
   <enumeration value="unqualified"/>
  </restriction>
 </simpleType>
 <simpleType name="ShapeTypeEnum">
  <restriction base="QName">
   <enumeration value="xshp:PointPropertyType"/>
   <enumeration value="xshp:PolygonPropertyType"/>
   <enumeration value="xshp:PolylinePropertyType"/>
   <enumeration value="xshp:PointMPropertyType"/>
   <enumeration value="xshp:MultiPointPropertyType"/>
  </restriction>
 </simpleType>
 <simpleType name="SimpleTypeEnum">
  <restriction base="QName">
   <enumeration value="integer"/>
   <enumeration value="date"/>
   <enumeration value="string"/>
   <enumeration value="boolean"/>
   <enumeration value="decimal"/>
  </restriction>
 </simpleType>
 <element name="annotation">
  <complexType>
   <choice minOccurs="0" maxOccurs="unbounded">
    <element name="appinfo">
     <complexType>
      <choice>
       <any namespace="##any" processContents="skip"/>
      </choice>
      <attribute name="source" type="anyURI"/>
     </complexType>
    </element>
    <element name="documentation">
     <complexType>
      <choice>
       <any/>
      </choice>
      <attribute name="source" type="string"/>
     </complexType>
    </element>
   </choice>
```

```
   <attribute name="id" type="ID"/>
   <anyAttribute namespace="##other"/>
  </complexType>
 </element>
</schema>
```

Appendix E

XMIF Meta-Schema

xmif4_1.xsm.xsd

```
<?xml version="1.0" encoding="UTF-8"?>
<schema xmlns="http://www.w3.org/2001/XMLSchema"
xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:xspxmif="http://www.xggt.org/xspxmif"
xmlns:xmif="http://www.xggt.org/xmif"
xmlns:xspxshp="http://www.xggt.org/xspxshp"
xmlns:xshp="http://www.xggt.org/xshp"
xmlns:xsm="http://www.xggt.org/mapping"
targetNamespace="http://www.xggt.org/xspxmif" elementFormDefault="qualified"
attributeFormDefault="unqualified">
 <annotation>
  <documentation>
 meta-schema defined in XML Schema.
  </documentation>
 </annotation>
 <annotation>
  <appinfo>
   <xsm:sourceSchema schemaLocation="../xshp/xshp4_1.xsd"/>
   <xsm:root name="xspxmif:schema"/>
   <xsm:mapping level="xsm:meta"/>
   <xsm:instanceSchemaImport namespace="http://www.xggt.org/xmif"
schemaLocation="xmifTypes4.xsd"/>
   <xsm:instanceSchemaImport namespace="http://www.xggt.org/xmif"
schemaLocation="xmifMeta4.xsd"/>
   <xsm:instanceMappingImport schemaLocation="xmifMeta4.xsm.xsd"/>
  </appinfo>
 </annotation>
 <element name="schema" xsm:source="xspxshp:schema">
  <complexType>
   <sequence>
    <element ref="xspxmif:import" maxOccurs="unbounded"/>
    <element ref="xspxmif:annotation" minOccurs="0" maxOccurs="unbounded"/>
    <element name="element" type="xspxmif:FeatureClass"
xsm:source="xspxshp:element"/>
   </sequence>
   <attribute name="attributeFormDefault" type="xspxmif:formEnum"
default="unqualified" xsm:source="@attributeFormDefault"/>
   <attribute name="elementFormDefault" type="xspxmif:formEnum"
default="unqualified" xsm:source="@elementFormDefault"/>
   <attribute name="targetNamespace" type="anyURI"/>
   <attribute name="version" type="token"/>
  </complexType>
 </element>
 <complexType name="FeatureClass" xsm:source="xspxshp:FeatureClass">
  <sequence>
   <element ref="xspxmif:annotation" minOccurs="0" maxOccurs="unbounded"/>
   <element name="complexType" xsm:source="xspxshp:complexType">
    <complexType>
     <sequence>
      <element ref="xspxmif:annotation" minOccurs="0" maxOccurs="unbounded"/>
      <element name="sequence" xsm:source="xspxshp:sequence">
       <complexType>
        <sequence>
         <element ref="xspxmif:annotation" minOccurs="0"
maxOccurs="unbounded"/>
         <element name="sequence" type="xspxmif:FieldSequence"
xsm:source="xspxshp:sequence"/>
```

```
        <element name="element" type="xspxmif:G_Field"
xsm:source="xspxshp:element"/>
       </sequence>
      </complexType>
     </element>
    </sequence>
   </complexType>
  </element>
 </sequence>
 <attribute name="name" type="NCName" xsm:source="@name"/>
 <attribute name="substitutionGroup" type="QName" fixed="xmif:_Feature"
xsm:source=""/>
 <anyAttribute namespace="##other" processContents="skip"/>
</complexType>
<complexType name="FieldSequence" xsm:source="xspxshp:FieldSequence">
 <sequence>
  <choice minOccurs="0" maxOccurs="unbounded">
   <element ref="xspxmif:annotation"/>
   <element name="element" type="xspxmif:Field"
xsm:source="xspxshp:element"/>
  </choice>
 </sequence>
</complexType>
<complexType name="Field" xsm:source="xspxshp:Field">
 <attribute name="name" type="NCName" xsm:source="@name"/>
 <attribute name="type" type="xspxmif:SimpleTypeEnum" xsm:source="@type"/>
 <anyAttribute namespace="##other" processContents="skip"/>
</complexType>
<complexType name="G_Field" xsm:source="xspxshp:G_Field">
 <attribute name="name" type="NCName" fixed="Geometry" xsm:source=""/>
 <attribute name="type" type="xspxmif:GeometryTypeEnum"
fixed="xmif:GeometryPropertyType" xsm:source=""/>
 <anyAttribute namespace="##other" processContents="skip"/>
</complexType>
<element name="import">
 <annotation>
  <appinfo>
   <xsm:text>
    <xspxmif:import namespace="http://www.xggt.org/xmif"
schemaLocation="xmifType4.xsd"/>
    <xspxmif:import namespace="http://www.xggt.org/xmif"
schemaLocation="xmifMeta4.xsd"/>
   </xsm:text>
  </appinfo>
 </annotation>
 <complexType>
  <sequence minOccurs="0">
   <element ref="xspxmif:annotation"/>
  </sequence>
  <attribute name="id" type="ID"/>
  <attribute name="namespace" type="anyURI"/>
  <attribute name="schemaLocation" type="string"/>
 </complexType>
</element>
<simpleType name="formEnum">
 <restriction base="string">
  <enumeration value="qualified"/>
  <enumeration value="unqualified"/>
 </restriction>
</simpleType>
<simpleType name="GeometryTypeEnum">
 <restriction base="QName">
  <enumeration value="xmif:GeometryPropertyType"/>
  <enumeration value="xmif:PointPropertyType"/>
  <enumeration value="xmif:RegionPropertyType"/>
  <enumeration value="xmif:PolylinePropertyType"/>
 </restriction>
</simpleType>
```

```
 <simpleType name="SimpleTypeEnum">
  <restriction base="QName">
   <enumeration value="integer"/>
   <enumeration value="date"/>
   <enumeration value="string"/>
   <enumeration value="boolean"/>
   <enumeration value="decimal"/>
   <enumeration value="float"/>
   <enumeration value="double"/>
  </restriction>
 </simpleType>
 <element name="annotation">
  <complexType>
   <choice minOccurs="0" maxOccurs="unbounded">
    <element name="appinfo">
     <complexType>
      <choice>
       <any namespace="##any" processContents="skip"/>
      </choice>
      <attribute name="source" type="anyURI"/>
     </complexType>
    </element>
    <element name="documentation">
     <complexType>
      <choice>
       <any/>
      </choice>
      <attribute name="source" type="string"/>
     </complexType>
    </element>
   </choice>
   <attribute name="id" type="ID"/>
   <anyAttribute namespace="##other" processContents="skip"/>
  </complexType>
 </element>
</schema>
```

Appendix F

The Generated XSHP-to-XMIF Schema Mapping Generator

xshp-to-xmif.xsm.xslt

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet xmlns="http://www.w3.org/2001/XMLSchema"
xmlns:fn="http://www.w3.org/2005/xpath-functions"
xmlns:xdt="http://www.w3.org/2005/xpath-datatypes"
xmlns:xmif="http://www.xggt.org/xmif"
xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:xshp="http://www.xggt.org/xshp"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
xmlns:xsm="http://www.xggt.org/mapping"
xmlns:xspxmif="http://www.xggt.org/xspxmif"
xmlns:xspxshp="http://www.xggt.org/xspxshp" version="2.0">
 <xsl:output xmlns="" method="xml" version="1.0" encoding="UTF-8"
indent="yes"/>
 <xsl:param xmlns="" name="isToTest" as="xs:boolean" select="false()"/>
 <xsl:template name="xsm:transferNamespace">
  <xsl:for-each select="/*[1]/namespace::*">
   <xsl:choose xmlns="">
    <xsl:when test="string(.)!='http://www.xggt.org/xspxmif' or
$isToTest=true() ">
     <xsl:copy-of select="."/>
    </xsl:when>
    <xsl:when test="string(.)='http://www.xggt.org/xspxmif' and
$isToTest=false() ">
     <xsl:namespace name="{name(.)}" select="
'http://www.w3.org/2001/XMLSchema' "/>
    </xsl:when>
   </xsl:choose>
  </xsl:for-each>
  <xsl:for-each select="document(&amp;quot;&amp;quot;)/*[1]/namespace::*">
   <xsl:copy-of select="."/>
  </xsl:for-each>
  <xsl:if xmlns="" test="not(/xs:schema/namespace::*[ string(.)=$_xsm_gvSns])
or /xs:schema/namespace::*[ string(.)=$_xsm_gvSns and name(.)!='' ] ">
   <xsl:namespace name="_tmp_From" select="/*[1]/@targetNamespace"/>
  </xsl:if>
  <xsl:variable xmlns="" name="_xsm_tnsRandom"
select="concat( &amp;quot;http://www.xggt.org/xspxmif&amp;quot;, '-' ,
generate-id() )"/>
  <xsl:namespace xmlns="" name="_tmp_To" select="$_xsm_tnsRandom"/>
  <xsl:attribute xmlns="" name="targetNamespace" select="$_xsm_tnsRandom"/>
  <xsl:attribute xmlns="" name="elementFormDefault" select="'qualified'"/>
  <xsl:attribute xmlns="" name="attributeFormDefault" select="'unqualified'"/>
 </xsl:template>
 <xsl:template xmlns="" name="xsm:mappingAnnotation">
  <xs:annotation>
   <xs:appinfo>
    <xsm:import schemaLocation="xmifMeta4.xsm.xsd"/>
   </xs:appinfo>
  </xs:annotation>
 </xsl:template>
 <xsl:template name="xsm:instanceSchemaImport">
  <xsl:element name="xspxmif:import">
   <xsl:attribute name="namespace">
    <xsl:value-of select="&amp;quot;http://www.xggt.org/xmif&amp;quot;"/>
   </xsl:attribute>
   <xsl:attribute name="schemaLocation">
```

```
    <xsl:value-of select="&quot;xmifTypes4.xsd&quot;"/>
   </xsl:attribute>
  </xsl:element>
  <xsl:element name="xspxmif:import">
   <xsl:attribute name="namespace">
    <xsl:value-of select="&quot;http://www.xggt.org/xmif&quot;"/>
   </xsl:attribute>
   <xsl:attribute name="schemaLocation">
    <xsl:value-of select="&quot;xmifMeta4.xsd&quot;"/>
   </xsl:attribute>
  </xsl:element>
 </xsl:template>
 <xsl:template name="xsm:metaMappingRootElementProcessing">
  <xsl:call-template name="xsm:transferNamespace"/>
  <xsl:if xmlns="" test="$isToTest">
   <xsl:attribute name="xsi:schemaLocation">
    <xsl:value-of select="&quot;http://www.xggt.org/xspxmif
xmif4_1.xsd&quot;"/>
   </xsl:attribute>
  </xsl:if>
  <xsl:call-template xmlns="" name="xsm:mappingAnnotation"/>
  <xsl:call-template name="xsm:instanceSchemaImport"/>
 </xsl:template>
 <xsl:variable xmlns="" name="_xsm_gvSns">
  <xsl:variable name="vsns" select="/*[1]/@targetNamespace"/>
  <xsl:choose>
   <xsl:when test="not($vsns)">
    <xsl:message terminate="yes">
     <xsl:value-of select=" 'No target namespace! Processing aborted.' "/>
    </xsl:message>
   </xsl:when>
   <xsl:otherwise>
    <xsl:value-of select="$vsns"/>
   </xsl:otherwise>
  </xsl:choose>
 </xsl:variable>
 <xsl:variable xmlns="" name="_xsm_gvSnsPrefix">
  <xsl:choose>
   <xsl:when test="not(/*[1]/namespace::*[ string(.)=$_xsm_gvSns]) or
/*[1]/namespace::*[ string(.)=$_xsm_gvSns and name(.)!='' ] ">
    <xsl:namespace name="_tmp_From" select="/*[1]/@targetNamespace"/>
    <xsl:value-of select="&quot;_tmp_From&quot;"/>
   </xsl:when>
   <xsl:otherwise>
    <xsl:value-of select="/*[1]/namespace::*[ string(.)=$_xsm_gvSns and
name(.)!=''] [1]/name(.)"/>
   </xsl:otherwise>
  </xsl:choose>
 </xsl:variable>
 <xsl:template xmlns="" match="/">
  <xsl:for-each select="xspxshp:schema">
   <xsl:call-template name="xspxmif:schema"/>
  </xsl:for-each>
 </xsl:template>
 <xsl:template name="xspxmif:schema">
  <xsl:element name="xspxmif:schema">
   <xsl:call-template name="xsm:metaMappingRootElementProcessing"/>
   <xsl:attribute name="attributeFormDefault">
    <xsl:value-of select="@attributeFormDefault"/>
   </xsl:attribute>
   <xsl:attribute name="elementFormDefault">
    <xsl:value-of select="@elementFormDefault"/>
   </xsl:attribute>
   <xsl:for-each select="xspxshp:element">
    <xsl:element name="xspxmif:element">
     <xsl:variable name="source" select="@name"/>
     <xsl:attribute name="xsm:source">
      <xsl:value-of
```

```
select="concat($_xsm_gvSnsPrefix,&quot;:&quot;,$source)"/>
    </xsl:attribute>
    <xsl:call-template name="xspxmif:FeatureClass"/>
   </xsl:element>
  </xsl:for-each>
 </xsl:element>
</xsl:template>
<xsl:template name="xspxmif:FeatureClass">
 <xsl:attribute name="name">
  <xsl:value-of select="@name"/>
 </xsl:attribute>
 <xsl:attribute name="substitutionGroup">
  <xsl:value-of select="&quot;xmif:_Feature&quot;"/>
 </xsl:attribute>
 <xsl:element name="xspxmif:complexType">
  <xsl:element name="xspxmif:sequence">
   <xsl:element name="xspxmif:sequence">
    <xsl:call-template name="xspxmif:FieldSequence"/>
   </xsl:element>
   <xsl:for-each
select="xspxshp:complexType/xspxshp:sequence/xspxshp:element">
    <xsl:element name="xspxmif:element">
     <xsl:variable name="source" select="@name"/>
     <xsl:attribute name="xsm:source">
      <xsl:value-of
select="concat($_xsm_gvSnsPrefix,&quot;:&quot;,$source)"/>
     </xsl:attribute>
     <xsl:call-template name="xspxmif:G_Field"/>
    </xsl:element>
   </xsl:for-each>
  </xsl:element>
 </xsl:element>
</xsl:template>
<xsl:template name="xspxmif:FieldSequence">
 <xsl:variable name="v">
  <xsl:for-each
select="xspxshp:complexType/xspxshp:sequence/xspxshp:sequence/xspxshp:elemen
t">
   <xsl:element name="xspxmif:element">
    <xsl:variable name="source" select="@name"/>
    <xsl:attribute name="xsm:source">
     <xsl:value-of
select="concat($_xsm_gvSnsPrefix,&quot;:&quot;,$source)"/>
    </xsl:attribute>
    <xsl:call-template name="xspxmif:Field"/>
   </xsl:element>
  </xsl:for-each>
 </xsl:variable>
 <xsl:choose>
  <xsl:when test="$v/*">
   <xsl:copy-of select="$v"/>
  </xsl:when>
  <xsl:otherwise/>
 </xsl:choose>
</xsl:template>
<xsl:template name="xspxmif:Field">
 <xsl:attribute name="name">
  <xsl:value-of select="@name"/>
 </xsl:attribute>
 <xsl:attribute name="type">
  <xsl:value-of select="concat(&quot;xs:&quot;, @type)"/>
 </xsl:attribute>
</xsl:template>
<xsl:template name="xspxmif:G_Field">
 <xsl:attribute name="name">
  <xsl:value-of select="&quot;Geometry&quot;"/>
 </xsl:attribute>
 <xsl:attribute name="type">
```

```
   <xsl:value-of select="&quot;xmif:GeometryPropertyType&quot;"/>
  </xsl:attribute>
 </xsl:template>
</xsl:stylesheet>
```