Translation Encoding for OGC Web Services

Master Thesis

by

Cai, Chunyuan

Master of Science in Geospatial Technologies

Erasmus Mundus

FEBRUARY 2010

Westfälische Wilhelms-Universität Münster (WWU), Institute for Geoinformatics (ifgi), Germany,
Universitat Jaume I (UJI), Castellón, Dept. Lenguajes y Sistemas Informaticos (LSI), Spain and
Universidade Nova de Lisboa (UNL), Instituto Superior de Estatística e Gestão de Informação (ISEGI), Lisboa, Portugal.
Submission Date: 2010-03-01

APPROVED BY

Dr. Sven Schade
Institute for Environment and Sustainability
European Commission - Joint Research Centre
Ispra, Italy
Email: sven.schade@jrc.ec.europa.eu

Prof. Dr. Werner Kuhn
Professor
Institute for Geoinformatics
University of Muenster
Weseler Strasse 253
D-48151 Münster, Germany
Email: kuhn@uni-muenster.de

Prof. Dr. Pedro Cabral
Professor
Instituto Superior de Estatística e Gestão de Informação
Universidade Nova de Lisboa
Campus de Campolide
1070-312 Lisboa - Portugal
Email: pcabral@isegi.unl.pt

ACKNOWLEDGEMENT

I would like to extend my sincere gratitude to all those who always support and accompany me throughout the completion of this thesis.

First, I want to thank my supervisor Sven Schade gratefully. He has showed great interest and support in my work, and always encouraged me to express ideas freely, all of those motivate me greatly in my research. What's more important, due to his professional instructions as well as great patience, I was able to conquer academic problems during my study. Moreover, I'm also grateful for the help of my co-supervisors Werner Kuhn and Pedro Cabral, without whom I couldn't finish my project so smoothly.

I heartedly thank Erasmus Mundus program for providing the opportunity of getting in contact with international researchers, as well as the chance of participating in various consortiums. And all kinds of presentations, as well as inspirational discussions make me grow fast in professional area and personality as well. Moreover, because of this program, I learn more about life from my new friends.

Finally, special thanks go to my beloved parents and my fiancée for their continuous support. Particularly, I want to thank my fiancée since she kept encouraging me and putting me back on track while I got confused.

ABSTRACT

Schema translation is an approach to achieve (geospatial) data harmonization. It affords a mechanism to make data described in one specific schema interoperate with others in heterogeneous schemas. Moreover, schema translation has been considered as an alternative to handle legacy (geospatial) data through translating original data into required ones. Schema translation therefore is also considered as a central role in context of INSPIRE (Infrastructure for Spatial Information in Europe). To achieve computer controllable schema translation, a variety of software products have already been developed based on different technologies. However, diversities in technologies and interfaces caused difficulties in the interoperation of schema translation in two levels:

1. Interoperation amongst existing schema translation solutions.
2. Interoperation between schema translation processing and standard OGC services.

Aiming at interoperations in these two levels, our objective is to present a comprehensively acceptable standard manner for application of schema translation. We propose a new concept *Translation Encoding*, which focuses on advancing an implementation specification and an encoding language for applying schema translation with standard OGC services.

We analyze three kinds of scenarios for translation encoding and decide *Provider-Translation* as our best practice for Translation Encoding implementation. To interpret translation contents into computer, we present an encoding language: *Translation Expression* and a corresponding software system design. At last, we develop a prototype to test our conceptual design. According to performance of the prototype, we conclude that our Translation Encoding solution achieves interoperating schema translation with OGC services.

Keywords: Schema Translation, OGC Services, Translation Encoding, Provider-Translation

CONTENTS

LIST OF ACRONYMS

API	**Application Programming Interface**
DTD	**Document Type Definition**
FME	**Feature Manipulation Engine**
GeoXSLT	**Spatially enabled XSLT**
GML	**Geography Mark-Up Language**
GUI	**Graphical User Interface**
HTML	**HyperText Mak up Language**
HTTP	**HyperText transfer Protocal**
INSPIRE	**Infrastructure for Spatial Information in Europe**
JAXP	**Java API for XML Processing**
KVP	**Key-Value Pair**
OGC	**Infrastructure for Spatial Information in Europe**
OOP	**Object Oriented Programming**
RCP	**Rich Client Platform**
RESTful	**Representational State Transfer**
RPC	**Remote Procedure Call**
SQL	**Structured Query Language**
UML	**Unified Modeling Language**
URL	**Uniform Resource Locator**
W3C	**World Wide Web Consortium**
WFS	**Web Feature Service**
WMS	**Web Map Service**
XLink	**XML Linking Language**
XML	**Extensible Mark-up Language**
XSD	**W3C's XML Schema language**
XSLT	**Extensible Stylesheet Language Transformation**

LIST OF FIGURES

LIST OF TABLES

LIST OF LISTINGS

1. Introduction

In Chapter one, we describe motivations about our research and accordingly define the central hypothesis and related research objectives. Flexible, interoperable computer controllable translation of geospatial data is our central aim. In context of translation, flexible and interoperability computer control require an intensive investigation into manners of schema-translatable system implementation, as well as the approach of cooperating it with other related systems. In the rest of this chapter, our underlying research methods are represented.

1.1 Scope and Problem Statement

Schema translation is an approach to achieve (geospatial) data harmonization. It affords a mechanism to make data described in one specific schema interoperate with others in heterogeneous schemas. Moreover, schema translation has been considered as an alternative to handle legacy (geospatial) data through translating original data into required ones. Correspondingly, schema translation plays a central role in context of INSPIRE (Infrastructure for Spatial Information in Europe) that geospatial data providers in Europe deliver their data content to the uniformed specifications. Moreover, schema translation can make effort to avoid possible resource waste caused by duplicated data collection and manipulation.

Realization of schema translation paradigm needs support of computer technologies. A variety of software systems have been developed already to achieve a computer controllable translation (Safe Software 2008, Snowflake Software 2008, Camp to Camp 2008 etc.). Technologies applied by these software products can vary. The diversity in technologies may accordingly cause the gap of cooperation between them.

Although people are aware of importance of standardization in schema translation, the process is not as easy as people image, especially before reaching a comprehensive agreement. Comparing to the standardization of schema translation in implementation, a standard manner about how to apply it may need more attention in advance. It can guarantee the cooperation of schema translation with other existing and broadly admitted specifications. The Open Geospatial Consortium (OGC) is leading the development of standards for geospatial services (OGC 2005) through developing and releasing a variety of documents (e.g. Web Map Service (WMS) Implementation Specification (OGC 2004), Web Feature Service (WFS) Implementation Specification (OGC 2005b) and Filter Encoding (OGC 2005c)), which are universally or practically supported by most popular software products (e.g. ArcGIS Server (ESRI 2009), GeoServer (GeoServer 2009), and Minnesota MapServer (MapServer 2009)). A mechanism for cooperating schema translation with these products has not been advanced yet.

Translation rule is a component which is adopted to describe the facets about how translation proceeds. Management of translation rules and execution of these rules have been identified as central requirements (Schade 2009). In a context of distributed computing, rules might be stored separately from translation execution services. There is little mechanism for rule reuse in different products. We summarize the limits of implementation of interoperation of schema translation as follows:

1. Interoperability between schema translation products breaks down, due to lack standardized manner for translation rule description and a unified interface specification.

2. Interoperability between schema translation products and other systems is limited because of lacking of comprehensive agreement of schema translation product implementation.

1.2 Objectives and Methodology

Considering the issues we mentioned before, we specify our research objectives. They are shown in the Figure 1.1:

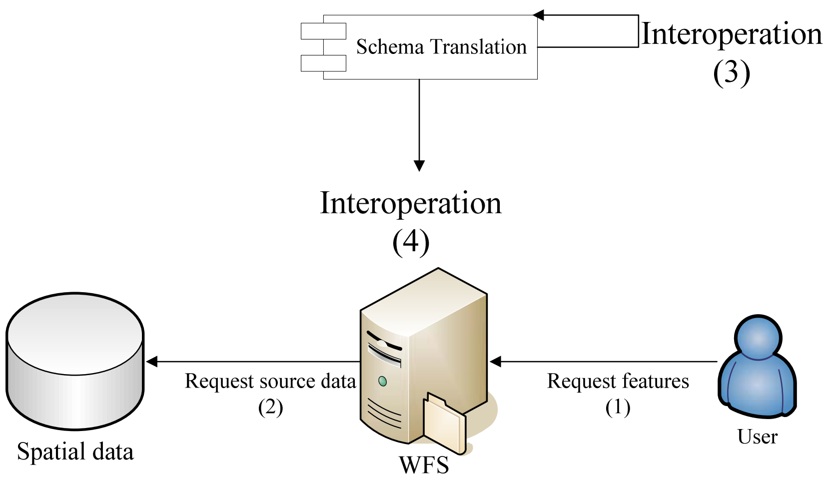


Figure 1.1: Architecture of objectives.

The lower part of figure shows a traditional manner of WFS service: the user sends a request to an implemented WFS product (Figure 1.1 (1)), and the WFS service requests source data to spatial data repository and encodes them into GML (OGC 2007) format (Figure 1.1 (2)), which is an encoding language specified by OGC in order to geospatial data transport and representation, and send it back to the user. The upper part presents an independent schema translation function. Although we use a standard *component* icon to represent it, this part can be various technical vendors: e.g. web services, standalone desktop applications, or an extension of other system.

Our objective is to present a comprehensively acceptable standard manner for application of schema translation. The manner should be able to achieve interoperation of schema translation in two levels: First, diverse schema translation products can work together (Figure 1.1 (3)); the other level guarantees the interoperation between translation products and OGC services (Figure 1.1 (4)). We choose one specific kind of OGC service (here refers to WFS) for our research. Through comparing the appropriate occasion of translation processing, we propose the architecture of system. After establishing the architecture, we present a new paradigm named *Translation Encoding* for OGC services. It is composed of a system implementation specification and a comprehensive language definition in XML.

This thesis deals mainly with architecture analysis and conceptual designing for interoperation of schema translation. The overall methodology of this thesis can be divided into three phases (Figure 1.2, left): namely literature review, conceptual development and implementation. In the literature review phase (Figure 1.2. left up), we will revise WFS specification and Filter Encoding Implementation (OGC 2005c), which restricts the returned dataset through setting up constrictions for data properties. A report of review is the expected outcome as necessary research background knowledge (Figure 1.2, right up). In the second phase, we will compare different manner of schema translation implementation and discuss the most efficient and appropriate architecture (Figure 1.2. left middle). The most crucial task: designing of Translation Encoding also happen in this phase. A best architecture for implementation and a language design for Translation Encoding is the result of this phase (Figure 1.2, right middle). Our research also includes a phase for system implementation (Figure 1.2, left bottom). A prototype will be used for testing conceptual model (Figure 1.2, right bottom).

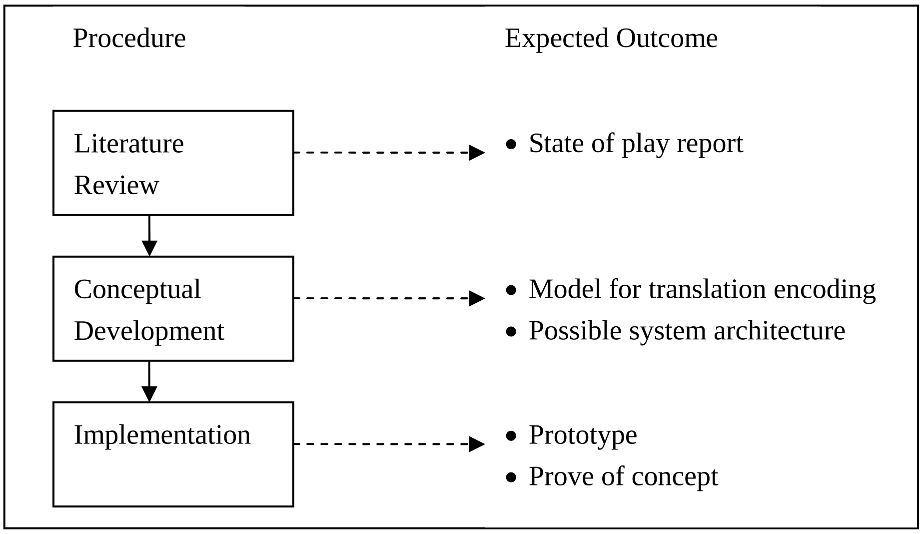


Figure 2.2: Overview of working steps and expected outcomes.

1.3 Intended Audience

This thesis addresses communities which have a research interest in GIS engineering and/or in implementing schema translation function. GI system designers are also to be intended, since they may be attracted by contents of practical system implementation. Although a deep understanding in web services and professional experiences are not mandatory, a common knowledge about basic principles for software design is helpful. Moreover, a boarding audience including GIS professionals and geographers may benefit from this thesis in the respect of geo data sharing.

1.4 Thesis Organization

The arrangement of chapters is as follows. Chapter 1 gives a brief introduction to the scenario and issue domain for our thesis work. Chapter 2 provides the background knowledge for this thesis, it describes the current situation of transformation for INSPIRE service (INSPIRE 2009), OGC service paradigm and related detailed WFS implementation specification, and a clarified definition of schema translation in our context (upper part of Figure 1.2). The third chapter analyzes and designs the architecture, after that it presents the design of the Translation Encoding. It interprets the architecture of the Translation Encoding along with a UML class diagram (middle part of Figure 1.2). Chapter 4 proposes a system design for implementing Translation Encoding extension with UML class diagrams, use case diagrams, as well as architecture figures rendering crucial components (also middle part of Figure 1.2). A prototype is also represented for testing conceptual design in this section. The last chapter provides a conclusion and discussion about the probable work in future. Appendix contains the examples of Translation Encoding expressions.

2. Background

In this chapter, we review three categories of concepts and paradigms as our research background knowledge: Namely, schema translation, OGC web service, and other computer science knowledge.

2.1 Schema Translation

Schema translation, as a well-established field in computer science, is a more complex concept and its semantics are poorly defined. Regardless of its muzzy name, a schema translation process actually transforms data content expressed in one schema into data content expressed in another, rather transforms the schema itself (Lehto 2007). We review some primary concepts about schema translation related to our research as follows.

2.1.1 Translation Types

The data translation can be roughly divided into three different levels, namely: a syntactic, schematic or semantic translation (Schade 2010). In our research, we need concern about the first two levels. While discussing about *a translation in the syntax level*, it refers to changing of the related schema definition language. A well-known example of this kind of process is to translate data stored in database into data represented in XML.

A schematic translation refers to modifying the structure and the schema vocabulary of data model. There are three kinds operations: one-to-one, one-to-many or many-to-many from source data to the corresponding entities in the target schema. The source and target schemas could easily be indicated by referring to the respective XML Schema (XSD) documents (W3C 2004). Normally, this translation requires a clear and standardized language to indicate the schema mapping. In our paper, we name the rule represented in this language as **translation rules**.

A semantic translation requires in-depth analysis to determine the semantic match between entities in the related schemas. Normally, it can not fully automate. Our research focus on a completely automate processing of translation in level 2. This kind is not included in our objectives.

2.1.2 Service Architecture of Schema Translation

In a web context, schema translation tends to be considered as an independent service. There are two traditional approaches (Lehto 2007) on how a schema translation service could be understood: *layered service architecture* and *chained service architecture*. The former one works as a layer in a stack and controlled by the server; and the latter one works as a link in a chain and correspondingly controlled by the client. The architecture for both options is shown in Figure 2.1. Our Architecture analysis is started off by these two solutions and then categorizes them according to a different scale.

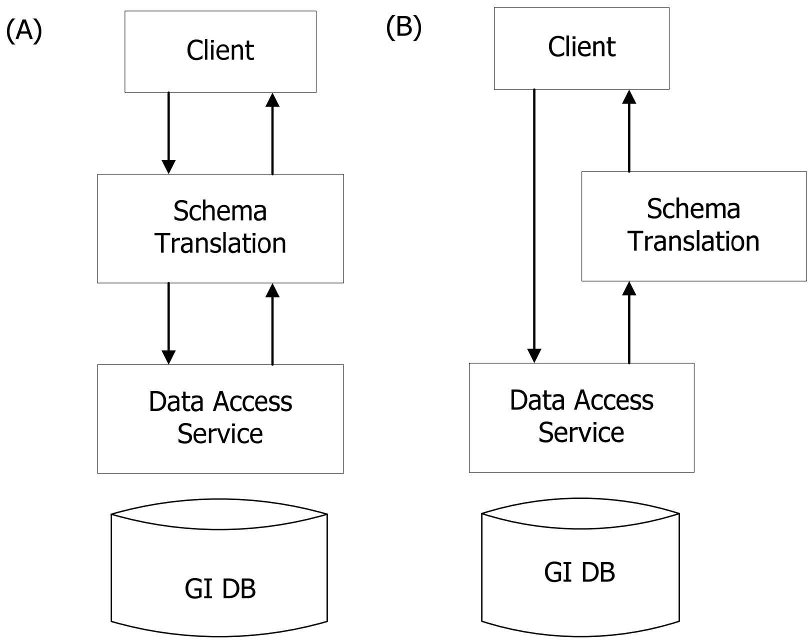


Figure 2.1: (A) Layered service architecture and (B) chained service architecture.

2.1.3 Schema Translation Example for INSPIRE

We present a schema translation Example for converting INSPIRE (INSPIRE 2004) data into NavLog (NavLog 2005) data. This example will be implemented as our prototype.

To achieve harmonization between geospatial data at different geographic scales and to avoid in duplicated information collection, the Infrastructure for Spatial Information in Europe (INSPIRE) is established in respect to environmental data within the European Union (INSPIRE 2004). INSPIRE proposes its recommendations for achieve existing spatial data infrastructure (Nebert 2004) share between various users. Thus, national data models must be mapped to INSPIRE Data Specification (INSPIRE 2009) also data sets need corresponding translation.

NavLog (NavLog 2005) is a data model (also known as data schema) in the forestry domain. It was developed for capturing, maintaining, and distributing forest road information in Germany. We focus on NavLog linear information for road feature, an English translation of attributes definition is shown in Table 2.1.

Table 2.1: Description of NavLog line attributes.

NavLog Attribute	Description
GEOMETRY	Center line of the road.
NAME	Name of the road.
LANEWIDTH	Restricted width of the roadway in meters (physical road width on the ground for trucks to pass).
CLEARWIDTH	width restricted by a clearance gauge in meters (visible width for driving).
CLEARHEIGHT	height restricted by a clearance gauge in meters (visible height for driving).
WAYCLASS	Classification of the road according to capabilities for wood removal (support for trucks with or without steering function).
CONCR	Pavement (road paved or not).
GRADIENT	Maximum slope of the road in percent.
BLOCK	Blockage (road blocked/closed).
COMMENT	Free text message.

All INSPIRE Data Specification adapts the Geography Markup Language (GML) in its last version 3.2.1 (OGC 2007b). Here we select elements of INSPIRE Data Specification related to NavLog line attributes for Transport Networks representation. The relevant attributes for INSPIRE are list in Table 2.2.

Table 2.2: Description of INSPIRE road link attributes.

INSPIRE Attribute	Description
centerlineGeomerty	Centre line of the road.
RoadName	Name of the road.
RoadWidth	Road width value.
FormOfWay	Classification of the road based on its physical attributes.
FunctionalRoadClass	Classification of the road based on its importance to the connectivity of the transport network.
RoadSurfaceCategory	State of the surface of the road (paved or unpaved).

Correspondingly, we need to make use of schema mapping to define the mapping relation for translating NavLog data into INSPIRE data. Elaborated matches are represented in Figure 2.2. The dash lines imply the mapping function needs further elaborations. This mapping relation will be applied as our prototype implementation.

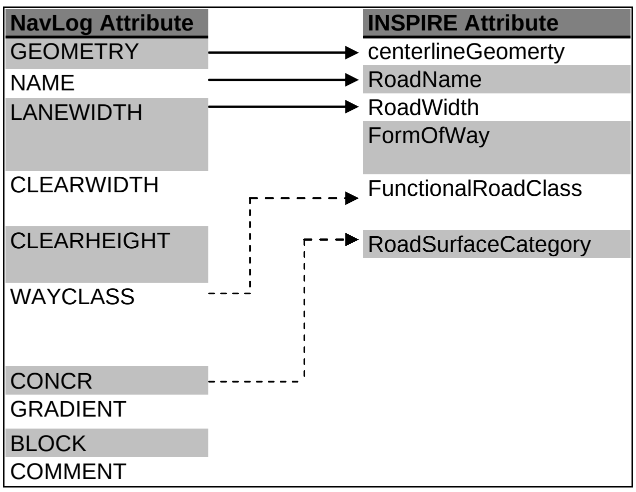


Figure 2.2: Identified schema mapping from NavLog to INSPIRE.

2.1.4 Translation Operations

In Figure 2.1 only simple one-to-one mappings are included. In general, more complex (one-to-many or many-to-many) mapping may happen. For instance, if the source data mode represents point information as a pair of separate attributes X and Y, according to INSPIRE data model which use a single attribute for point, the translation should be a two-to-one mapping.

Considering the fact normally source data mode can not mach target data perfectly, translation operation may be required in this context. For instance, some data mode has the information about vertex of road, but target data model imports a new attribute: *length of road*. The translation operation can be adopted for numeric calculation. Categorizations of the large amount of translation operations have been achieved previously (Lehto 2007; Schade 2010). According to Schade (2010) renaming is the simplest category, and filtering, reclassification, value conversion, augmentation, merging and morphing are more complex.

After analyze the source and target scheme and defining mapping rules, source data must be retrieved and translated based on the rules. The translated target data must be delivered correspondingly.

2.2 OGC Web Services

In this section, we review some knowledge about OGC services according to a variety implementation specification proposed by OGC and other related standards. Our research is aiming at advancing an implementation specification and an encoding language for OGC. Thus, an intensive investigation into existing OGC standards is necessary. It guarantees that our solution can follow OGC manners and work with existing OGC standards without conflicts.

2.2.1 GML

GML, the abbreviation of Geography Markup Language, is an XML encoding language for the transport and storage of geographic information modeled according to the conceptual modeling framework (OGC 2007b). GML language structure is defined in the XML Schema Document (XSD) syntax (W3C 2004), which is normally considered as an XML-based alternative to DTD and applied to describe the structure of an XML document (W3schools 2009). Moreover, GML supports for describing geospatial application schemas for specialized domain and information communities. A GML application schema is an XML schema document based on the GML rules for specific application, and defines a vocabulary of geographic objects for a particular domain e.g. application schema for earth observation (OGC 2007d). We limit our schema translation in schematic translation level (2.1.1). It implies that the translation is to aiming at translation GML from one application schema into another one.

2.2.2 WFS

WFS offers a capability to create, modify, delete and query feature instances from an underlying data storage. These operations are invoked using any HTTP-transportable distributed computing platform to send a request message to a service, which then processes the request and generates a response message. There are three main operations to render the landscape of WFS: GetCapabilities, DescribeFeatureType and GetFeature (Scott 2009). GetCapailities is a standard operation specified in OGC Web Services Common Specification (OWSs) (OGC 2007c). This mandatory operation allows any client to retrieve metadata about the service capabilities provided by any server. GetFeature operation practically processes the retrieving feature instances and return data encoded in GML (OGC 2007b). A normally neglected operation: DescribeFeatureType can be used to obtain detailed metadata for feature layer. Since this operation describes the detailed and comprehensive structure of feature layers in XML schema, it is normally invoked before the GetFeature operation in order to obtain a clear reference for setting constraint to return data set. A standard procedure for Basic WFS is started off by requesting GetCapabilities to get description of service capability. And then, users directly apply GetFeature to retrieve feature instances. Only when users need a comprehensive schema description for retrieved data, they will intend to invoke DescribeFeatureType operation.

According to the WFS implementation specification, the concreted WFS can be roughly categorized into three types: Basic WFS, XLink WFS and Transaction-WFS. Basic WFS shall implement the GetCapabilities, DescribeFeatureType and GetFeature operations, thus it is functionally considered as a READ-ONLY web feature service (Scott 2009). And it returns the feature instances encoded in GML. For Translation-WFS, it provides a capability of re-write data to services. The Basic WFS is considered as our research hypothesis.

2.2.3 Filter Encoding

A filter expression is a construct used to constrain the property values of an object type for the purpose of identifying a subset of object instances (OGC 2005c). The filter encoding is a common component that can be used by a number of OGC web services. It is represented in XML encoding language. Normally it is used with other OGC service requests in HTTP POST method (2.3.1). Filter Encoding needs a computer controllable component. It can convert filter expression into standard SQL *Where* clauses in order to setting constraints for return data sets. Our translation encoding should work with filter encoding without conflicts. In language design, we need to avoid applying language elements with the same name as filter encoding language.

2.3 Related Computer Science Knowledge

Computer science is not our research field. However, to understand our implementation design, he knowledge about HTTP, web services, design pattern is helpful.

2.3.1 HTTP

HTTP is a standard protocol of Internet. It standardizes the manner hypertext transport in internet environment (W3C 2009). It defines different methods for different purposes of message manipulation. Normally, OGC services support two HTTP methods for retrieving feature instances: GET and POST (Scott 2009). GET method deliver a pair of parameter specify in Key-Value pair (KVP), and POST envelop the parameter information within HTTP message body. The HTTP methods should be specifically distinguished according to the purpose of communicating services (Fielding 2002), and GET method mostly tends to be applied in retrieving some data (Leonard and Sam 2007). If a WFS request only need to specify a few of parameters, GET method should be appropriate in this case. Whereas, the requests need complicated encoded structure, for instance, filter encoding expression in XML format, the POST become the feasible HTTP method. Assigning of parameters in GET method with morn than 255 characters is not recommended (W3C 2009).

2.3.2 Web Service

Web Service is proposed in order to offering a mechanism which achieves cooperation of functions implemented in different platforms (Martin 2009). It is defined as a stateless and self-contained function and published in web. Web service must be searchable and executable through web environment. It is naturally a peer-to-peer message exchange and communication. One peer is called *consumer*. It requests services or functions form the other peer, namely *provider*. OGC web services are technically considered as web services (Nicolai 2008).

Web Service can be implemented in two manners: Resource-Oriented also known as RESTful services and RPC-style (Leonard and Sam 2007). OGC services are mostly implemented as the latter one, which implies that the service does not comprehensively distinguish HTTP Method; instead through specify URL parameter depicts necessary operations.

2.3.3 Design Pattern

Design Pattern is presented for design of reusable components, and it is the basis of object oriented programming (OOP) (GOF 1995). It is represented in 23 kinds and categorized into three primary kinds: creational pattern, structural pattern and behavioral pattern. Although in practical system design, it is not necessary to implement all kinds, our implementation can benefit from many of them. Notably, one pattern named *observer* needs to be focused in our research. It guarantees the schema translation can be notified to activate and automatically executed with system statues changing.

3. Translation Encoding Analysis and Design

In this section, first we propose a new concept *Translation Encoding* based on a requirement analysis, second we analyze three kinds of scenarios for translation encoding application in terms of three aspects: participant, architecture and encoding design. Participant indicates the involved system roles and their responsibility. Depending on different architecture design, participants vary. A discussion for our best practice for Translation Encoding implementation is presented as the summary of this chapter.

3.1 Requirements for Translation Encoding

Nowadays, a variety of successful software products for schema translation emerged e.g. Feature Manipulation Engine (FME) (Safe Software 2008), GeoXSLT (Klausen 2006), model-driven WFS (Donaubauer, Fichtenberger et al. 2006). Meanwhile, due to applying distinct strategies for translation rule definition, specific rule files can not be understood by other platforms. Specific platform can only understand its own mapping rule. It causes a tight coupling between rule language and concreted translation processing. For instance, FME, as a popular and successful commercial translation product, proposes a strategy to transform original state to destination state using mapping file named “transformation specification”. This rule does completely depend on the translation capability offered by FME and it can not be reused in other platforms except for FME itself. It causes the barrier of interoperation among different products, and correspondingly limits the later reuse.

Another facet of applying schema translation should be improved with the bloom of the distributed computing. With the revolution of web techniques, resources can be stored separately from concrete services. System data and function processing are able to be respectively published independently. Integrating resource and service in the real time can achieve in the flexible assignment of tasks. For example, users may retrieve any kinds of translation rule files from a rule repository and then deliver it to executable service automatically without knowing the details of rules or translation processing details. Until now, due to lack a standard for implementation rules or translation tools, users have to choose translation solution manually and be used to running it in a standalone environment.

Our research would neither extend the existing product group, nor provide a new technical solution to replace any existing one, whereas we would propose a mechanism which can binding translation products and appropriate rules in real time. This mechanism, we name with a new concept: *Translation Encoding*. We present this new concept and agree that it is applied for enhancing the OGC services with the schema translation capability. Functionally, Translation Encoding should involve some key technical requirements:

1. Any OGC service that requires the translation capability can make use of the Translation Encoding expression with it.

2. Translation Encoding would automatically lookup and then selects the appropriate schema translation solutions (platforms or services online).
3. Translation Encoding includes a strategy for defining general translation rules that can be understood by any involved translation solutions.
4. Translation Encoding would be able to run translation processing. It is more difficult that it seems to be. For instance, in the context of an standalone translation software, the translate product can be activated by a simple command, whereas in the context of services, some technical details e.g. binding services, managing message states, determining message exchange pattern, selecting the protocol for message transportation etc. have to be considered.
5. In the architecture aspect, translation processing should be designed as a loosely coupled component. It implies that schema translation must be conveniently added or removed from OGC service without influencing original construction.
6. Translation Encoding must have capability for exception handling.

Translation Encoding includes two main tasks in different aspects. One, which is considered in system implementation aspect, needs to present architectures for translation interoperation with OGC services; the other one is an encoding language design, which plays as the vector of instructions for translation. We call it *Translation Expression*. A Translation Expression is a construct used to express translation contents. In this chapter, we concentrate on the Translation Encoding task 1: analyzing implementation architecture. Expression design for the first scenario is also presented here. The detailed expression design for our best practice will be interpreted in the next chapter.

3.2 Scenario Analysis

In our research, we discuss three scenarios according to their possible manners: *Schema Translation working as Extended Function*, *Schema Translation Working with Customer*, *and Schema Translation Working with Provider*.

For each scenario, we will discuss it beginning with participant analysis. OGC web services, which are technically based on web service, naturally have two traditional participants: a web service and a client, also known as the *provider* and the *consumer* respectively. In terms of OGC services, taking WFS as an example, the participant, who offers the access which allows users to retrieve feature objects through service interfaces, can be considered as provider, and correspondingly users of geo-data is the consumer. Along with involving a capability of Translation Encoding, the structure of roles may become ambiguous. Concerning about the possibility of increasing complexity of system while importing new participants, reviewing the system roles and their relations will be the crucial task in the beginning.

Architecture for implementation is proposed for all kinds of scenarios classified above. For each category, the architecture is composed of an architecture diagram, a detailed description of workflow, as well as a discussion of recommendatory manners for each participant: for instance whether consumer is a standalone desktop system or distributed computing platform etc.

Language design of Translation Expression for Translation Encoding is the core task of our research. However it would not be appropriate for all possible scenarios. The second scenario may require other approaches instead of applying Translation Expression.

3.2.1 Schema Translation Working as Extended Function (Scenario 1)

An intuitive idea to engage the schema translation into OGC services is directly extend the existing Services' capability. In this scenario, OGC Services are implemented not only depending on its specification, but also extended with an extra function: schema translation.

Participant

There is no new participant is necessary for this scenario, thus the provider-consumer structure will be reserved. For instance, the source data is presented in NavLog schema and offered through WFS provider. With the help of translation processing, the WFS can offer not only NavLog data, but also data in other schemas depending on translation capability. Since the internal structure of WFS is immutable, we reserve roles of a web service: the consumer and the provider in this scenario. However, the provider needs to be extended with a translation capability. Thus the role structure is not changed, but the provider responsibility is extended.

Architecture

For this scenario, the most primary evolution occurs while implementing services. Translation processing should be activated with original service requests. Hence, schema translation is implemented internally as an extended function of WFS service. Comparing to traditional WFS service, the improved provider reserves all capability of original WFS; in addition it is given a new capability to translate data from one schema into another one. The supported schema depends on schemas which translation component can process.

A standard Basic WFS implementation provides schema description through an operation named "DescribeFeatureType". This operation represents data schema in XML schema according to the application schema applied in source data. For instance, if one specific source data stored in database are described in NavLog, this operation will use XML Schema of NavLog as response. In our case, schema description should be extended in order to offer a list of data models which can be translated into, instead of merely describing model specified by original spatial data. For instance, while our WFS has the ability to convert data from NavLog into INSPIRE, the schema description for the source data motioned before should involve one new schema for the same data content. An extended parameter, which specifies the schema model for output, should be sent with GetFeature request. Hence we need import one new operation for describing supported schema and extend GetFeature operation.

Inspired by DescribeFeatureType which offers an XML schema to describe the original data model, we can implement another access: *GetFeatureModelList*. Comparing to operation DescribeFeatureType, GetFeatureModelList depicts a list of all schema supported by translation processing, rather than source data models. GetFeatureModelList is not responsible for providing a detailed description of schemas; instead it only offers a list of the schema name and related links. For users who need details of each schema, they can still apply DescribeFeatureType.

For traditional WFS consumers who do not need schema translation, they can still follow the workflow specified in standard WFS specification. In the other hand, for those who needs translation schema, checking supported data model through access *GetFeatureModelList* will be the first step. Moreover, specifying the target schema model with GetFeature request follows. The architecture is shown in Figure 3.1.

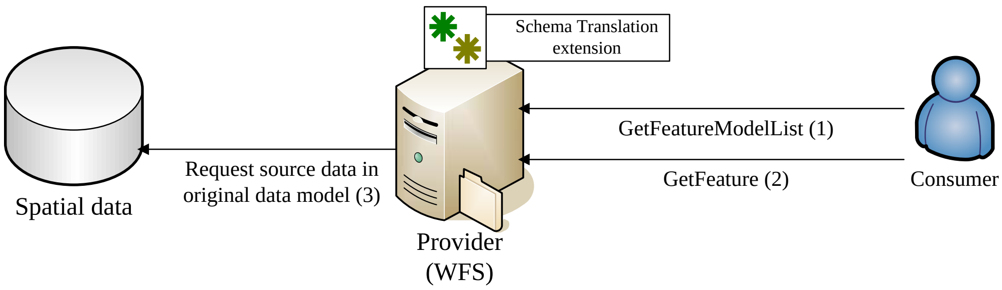


Figure 3.1: Architecture of scenario 1.

The sequence of workflow is described as follows:

1. Consumer requests for a data model list supported by the provider through GetFeatureModelList operation. The provider offers the list of supported data schema according to the translation capability.

2. Depending on the result of step 1, consumer selects the needed schema and specifies it through the parameter *schemaType*. This parameter will be sent with GetFeature operation.

3. The provider requests source data to spatial data repository and internally execute translation function according to specified parameter schemaType.

This translation would be either syntactic or schematic translation. It depends on the translation extension. If translating source data into target data schema before GML encoding, it will be former type; but if it executes translation after encoding source data, it would be a translation XML to XML. The architecture can be categorized into layered service architecture (2.1.2)

Translation Expression Design

We recommend this scenario apply GET method for translation expression depiction, because users only need to send one simply operation: *GetFeatureModelList* and one parameter: *schemaType* with GetFeature operation. In this case a KVP (Key-Value Pair) encoded (2.3.1) is easier that XML encoded for this scenario. However, in order to offer a comprehensive design we also offer an XML encoding design (shown in List 3.1) for this scenario. The XML encoded expression tends to be applied with POST method.

A *GetFeatureModelList* element includes zero or one attribute. If there is no attribute specified, then it would be interpreted as requesting a description of all models that a WFS can service. The following XML schema fragment defines the XML encoding of a GetFeatureModelList request (Listing 3.1).

```
<xsd:element name=" GetFeatureModelList " type="GetFeatureModelList "/>
<xsd:complexType name="GetFeatureModelList">
<xsd:complexContent>
    <xsd:extension base="wfs:BaseRequestType">
        <xsd:attribute name="numberOfList"    type="xsd:integer" use="optional" minExclusive ="0" />
    </xsd:extension>
</xsd:complexContent>
</xsd:complexType>
```

Listing 3.1: Translation Encoding of request of scenario 1.

This element contains an optional attribute: *numberOfList*, which is used to indicate the amount of data schema in response. The minExclusive value of 0 indicates that request has to specify a number more that 0 for the number of model. If the amount of translatable data schema is less than the specified number, this attribute will be ignored.

In response to a GetFeatureModelList request, a WFS implementation must be able to present a XML document describing supported models. The corresponding XML schema fragment is defined in Listing 3.2.

```
<xsd:element name=" ResponseGetFeatureModelList " type="wfs: ResponseGetFeatureModelListType "/>
<xsd:complexType name=" ResponseGetFeatureModelListType ">
    <xsd:sequence>
        <xsd:element name="Model" type="wfs:ModelType" minOccurs="0" maxOccurs="unbounded"/>
    </xsd:sequence>
    <xsd:attribute name="numberOfList"    type="xsd:integer" use="required" minInclusive ="0" />
</xsd:complexType>
<xsd:complexType name=" ModelType ">
      <xsd:attribute name="modelName"    type="xsd:string" use=" required " minOccurs="1" maxOccurs="1"/>
        <xsd:attribute name="schemaLink "    type="xsd:string" use="required" minOccurs="1" maxOccurs="1"/>
</xsd:complexType>
```

Listing 3.2: Translation Encoding of response of scenario 1.

A XML element: *ResponseGetFeatureModelList* presents all supported data models. It can contain zero or more *Model* element, which is comprised of two required attributes: *modelName* and *schemaLink*. They respectively indicate the name of data model and link to XML schema. Additional attribute: *numberOfList* indicates the number of supported models. It will be useful while the requested numberOfList is less that practical amount.

A parameter specifies the data schema for target data with GetFeature. It is defined in GetFeature encoded and a possible XML schema fragment is shown as follows:

```
01<xsd:attribute name="schemaType"    type="xsd:string" use=" optional " minOccurs="1" maxOccurs="1"/>
```

Listing 3.3: Translation Encoding of schemaType.

This attribute can be considered as a trigger for activating translation function and only appear once in a GetFeature operation. Moreover, it must work with GetFeature operation. The response strictly follows the standard GetFeature response. An example including all defined elements is shown in Appendix I.

3.2.2 Schema Translation Working with Customer (Scenario 2)

This scenario separates translation processing as a separated function. Activating and controlling translation is controlled by consumer. It is named as *Consumer-Translation*. Comparing to scenario 1, Consumer-Translation separates translation function from provider and makes users to decide the occasion for activating translation processing. However, it can not be considered as chained service architecture (2.1.2). In this scenario, translation function is independent on provider or consumer systems and needs to be activated manually by consumers. Technically, it lives in a totally independent progression and does not share computing and storage resources with the provider.

Participant

In this scenario, in addition to consumer and provider, a new role named *assistant* is involved, who will take the responsibility of translation processing. According to the occasion of invoking translation function, assistant can be named: *consumer-assistant*. Notably, comparing to the provider defined in the first scenario, the new provider reverts to the traditional web service provide. It implies that provider is an ordinary WFS service without translation capability.

Architecture

Comparing to scenario 1, this scenario postpones the occasion of translation processing until users retrieve data from providers. Technically, this mode allows users to choose translation products either running as standalone software, or functions published in a context of distributed environments e.g. a web service.

The Architecture actually is composed of two sub-systems: one is an ordinary WFS implementation; the other is an independent system for schema translation. The provider (referring to OGC services) is not conscious of the existence of translation function, and vice versa. The interoperation of translation function and OGC service need to be manually controlled by users. An architecture based on this mode is shown in Figure. 3.2.

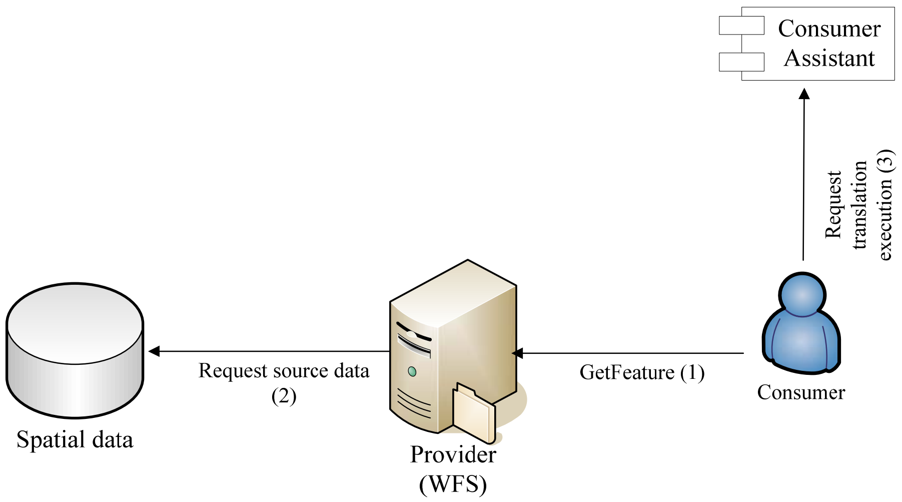


Figure 3.2: Architecture of schema translation with Consumer.

First consumer retrieves encoded geospatial data form a WFS server. After the retrieving the data, the consumer needs to determines whether to invoke a translation function. If translation is needed, the consumer sends the retrieved data to a concrete translation product and later obtains translated data. Comparing to the scenario 1 in Figure 3.1, the extension of translation is replaced with a separate platform. The data retrieved from provider is already XML encoded in GML form, and the translation occurs in the schematic level. The detailed workflow is summarized as follows:

1. Consumer requests encoded data through GetFeature operation.
2. The provider requests source data to spatial data repository and execute translation in syntactic level: translate data in database into XML encoded data (GML).
3. Consumer delivers encoded data to consumer-assistant to execute a schematic translation and get target data as result.

Notably, although we use a component icon to represent consumer-assistant, it can be any kind of implementation running in a separate platform e.g. an isolated desktop application, or an independent web service.

Translation Expression Design

A possible disputation about whether Translation Encoding to be integrant while applying the consumer-translation should be clarified. If we only concern it in system aspect, consumer -translation is a combined system including two separate sub-systems: traditional OGC services and schema translation system which respectively runs in their own isolated progressions. Regarding consumer-translation structure, Translation Encoding will more concentrate on the architecture design rather than translation expression design. Thus, we consider Translation Encoding expression to be not necessary in this case.

3.2.3 Schema Translation working with Provider (Scenario 3)

To achieve automatic translation processing, and also to guarantee the loose coupling of system structure, we propose another scenario: *Provider-Translation*.

Participant

Comparing to scenario 2, the consumer-assistant is replaced with a new role: *provider-assistant*: who is more related to providers and directly controlled by them. Consumers can indirectly communicate with it though the mediator which is embedded in a provider.

Architecture

From the literal sense of this scenario, the occasion of translation process occurs before data arriving at consumers. However, it dose not simply shift the translation processing to an earlier phase. It requires WFS service can reply users' request for translation processing, as well as search and bind a concrete translation service in run time. Different from the strategy of scenario 1, the translation function is considered a separate platform, rather than an extension. It guarantees the existing translation products can be involved into.

For Provider-Translation, the translation function is better to be implemented as a standard web service, rather than a standalone desktop application. In this case, the translation function, also called as provider-assistant, should be uniformed into a standard interface, which makes OGC services not necessary to consider about differences caused by heterogeneous interfaces and diverse translation rules specified by various provider-assistants.

In this mode, consumer should firstly check the source data schema with standard DescribeFeatureType operation, and determine whether to activate the translation function with GetFeature operation. Consumers are allowed to specify the concrete translation services for executing translation. If consumers already have the translation rule, the architecture should support for submission of customized translation rule and executes translation based on it. The suggested architecture is shown in Figure 3.3.

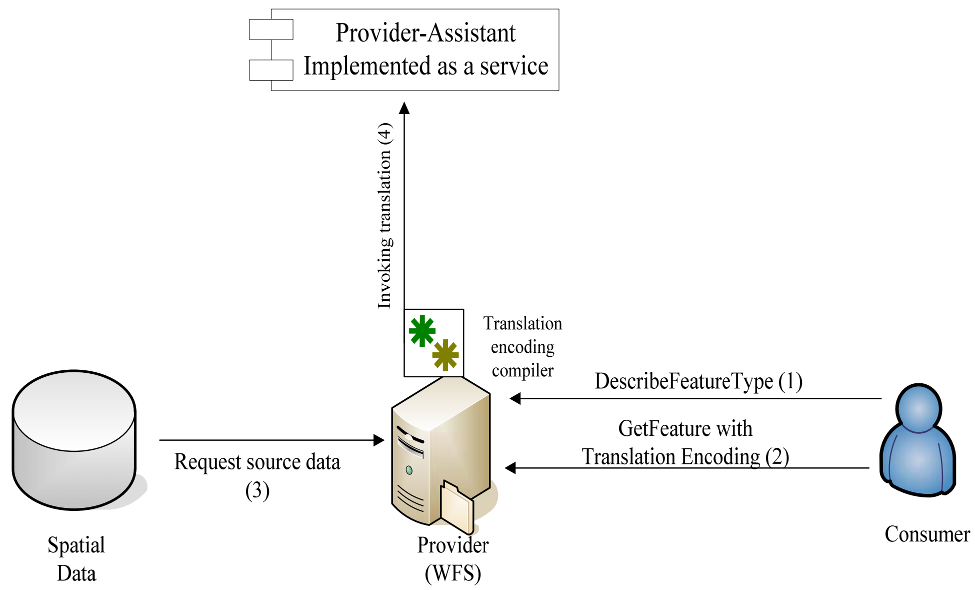


Figure 3.3: Architecture of schema translation working as Provider-Translation.

We do not intend to add a new operation for activating translation processing; instead the instruction for activating translation processing is enveloped with a GetFeature operation. Notably a new extension named *Translation Encoding complier* needs to be developed. Functionally, it extracts Translation Encoding expression fragment from GetFeature operation and then interprets the instructions to system. The detailed workflow is described as follows:

1. Consumer requests for the source data schema with DescribeFeatureType operation. The provider offers the result according to the source data model.

2. According to the result from pervious step, Consumer determines whether to activate the translation function. If consumer needs an additional translation processing, A GetFeature with Translation Encoding expression will be sent to a provider. Otherwise, applying a standard GetFeature operation.

3. The provider requests source data to spatial data repository.

4. Based on the Translation Encoding instructions enveloped within GetFeature operation, Translation Encoding complier notifies the provider to choose an appropriate translation product. The provider in turn makes a binding to the translation service in real time in order to invoke translation processing. After translation processing, provider sends the target data back to the consumer.

Regarding the submission of customized translation rule, we do not expose a specific operation. Instead, we require use to specify resource link to rule files in Translation Encoding expression.

Although we recommend web service techniques for implementation of provider-assistant, technically it can also be an isolated program. Furthermore, because this encoding will be a more complicated XML fragment comparing to scenario 1, we suggest applying HTTP POST method as the communication method.

Translation Expression Design

The detailed translation expression design of this scenario will be presented in 4.1 as our best practice for Translation Encoding.

3.3 Summary and Discussion

In this chapter, we discuss three scenarios for application of schema translation. We define the participants for each scenario, which is shown in Table 3.1.

Table 3.1: Summary of participants of Translation Encoding.

<table>
<tr><td rowspan="5">Scenario 1</td><td colspan="3">Schema Translation working as an extended function</td></tr>
<tr><td rowspan="3">Participant</td><td>Name</td><td>Description</td></tr>
<tr><td>Consumer</td><td>Users of OGC Services</td></tr>
<tr><td>Provider</td><td>Service Providers of OGC Services including schema translation function</td></tr>
<tr></tr>
<tr><td rowspan="5">Scenario 2</td><td colspan="3">Schema Translation working with Customer</td></tr>
<tr><td rowspan="4">Participant</td><td>Name</td><td>Description</td></tr>
<tr><td>Consumer</td><td>Users of OGC Services</td></tr>
<tr><td>Provider</td><td>Service Providers of OGC Services</td></tr>
<tr><td>Consumer-Assistant</td><td>Schema Translation function controlled by client</td></tr>
<tr><td rowspan="5">Scenario 3</td><td colspan="3">Schema Translation working with Provider</td></tr>
<tr><td rowspan="4">Participant</td><td>Name</td><td>Description</td></tr>
<tr><td>Consumer</td><td>Users of OGC Services</td></tr>
<tr><td>Provider</td><td>Service Providers of OGC Services</td></tr>
<tr><td>Provider-Assistant</td><td>Schema Translation function controlled by OGC Services</td></tr>
</table>

Comparing to scenario 1, we argue that separate assistant role from provider in scenario 2 and 3 makes sense. It presents a flexible manner for integrating translation processing into OGC services. According to the controlling of translation processing, the provider can be classified into consumer-assistant and provider-assistance respectively with scenario 2 and 3.

Regarding architecture presented before, we discuss their advantages and disadvantages to determine our best practical approach for Translation Encoding.

For scenario 1, the advantages are summary as follows:

1. It is simple and easy to be implemented.
2. Users do not need to concern about translation rules. They just need to notify service of the data schema they want for the target data, the service will process translation internally and it would be opaque to users.
3. For developers of services, they can freely design any strategy for translation. It implies the developers can freely decide the structure of translation rules.

However, several disadvantages should be noticed:

1. Since the implementation of translation function is totally determined by developers, for users, they lose the chance to select the appropriate translation products, rather subject to developers' choice.

2. For this scenario, it does not expose an access to user for extending the translation function. Thus users can not add new translation operations if they fail in finding needed translation.

3. This scenario does not offer a way for uploading translation rule by users. Because the translation product has already determined in developing phase. For specific rules submitted by different users, the product can not understand.

To avoid in the disadvantage in scenario 1, we propose scenario 2, and the advantages are summarized as follows:

Regarding interoperation and reusability this Customer-Translation is better than scenario 1:

1. The translation function is totally independent of WFS service, so consumer can freely choose diverse existing translation products. On the other hand, WFS service does not need to be modified.

2. The consumer is able to determine whether to execute translation and the occasion for executing translation.

However, in the technical aspect, this solution is a combination of two heterogeneous systems. Interoperation just happens in business scope, rather than in technical scope. It brings new issues:

1. The processing needs twice communication with different services, if the translation function is a service in internet. Under the internet environment, the connection normally is time and resource consuming, especially while transporting GML data in the capacity in megabyte level. The performance has to decline with the increased communication.

2. When the translation function is deployed within a same system as consumer, transferring data in internet will occur once. Nevertheless, the system has to burden the storage of double even triple amount of data. Another option is to develop an additional program to manage the data in the different schema.

3. In the technical aspect, this strategy does not implement an automatic interoperation. The activating of translation function has to depend on manual manipulation.

Since the execution of translation is manually controlled by system users, users can invoke it through a web service interface, a web page or a desktop GUI at any occasion. But before they know the existence of translation function, the interoperation would not be possible. Technically, there is little internal relation between WFS service and translation function. So we do not recommend to implementing a encoding for this mode.

Scenario 3: Provider-Translation introduces a solution for integration translation processing published as separate products. It guarantees the loose coupling between translation processing and OGC service processing. Comparing to scenario 2, it offers a mechanism to make system control translation implicitly. Moreover, users can freely activate or inactivate additional translation functionality through encoding expression. Rule files can be restored in an external system and be researched in distributed system for reuse.

Even though, it still can not address the issue about multiple-connection. It implies that to obtain target translated data, multiple internet connections are necessary: one mandatory for retrieving WFS data, one for translation processing in the context of translation implementing as service products. However, compare to scenario 2, Provider-Translation has prior advantage in interoperation: it is also our first objective for this thesis, so we suggest this scenario as our best practical approach.

As a conclusion, we recommend scenario 3: Provider-Translation as our best practical approach for implementation. In the next chapter, we present detailed encoding language design for this mode and a system design for implementation of software product. The design should be vendor independent. It implies that we intend to avoid any possible restriction caused by specific properties and features of language. We propose our implementation in a generic manner. Tow kinds of UML diagrams (Grady et al. 2005) will be applied for the design task.

4. Language and System Design

Regarding our best practice Provider-Translation, this chapter introduces an encoding language design and a corresponding system design. For the former design task, we begin with an explanation for reasoning of using XML as encoding language and then a comprehensive language structure is presented in XML schema. In terms of system design, we start off by analyzing required functions through reviewing the architecture of Provider-Translation. After that, overall system design and component design are represented in detail. At last, a prototype is rendered for design testing.

The Unified Modeling Language (UML) has been working as an industry standard for visualizing, specifying constructing software system (Grady et al. 2005). Our system design, including overall and component design, will be represented in two kinds of UML diagrams: user case diagram and class diagram. The former is normally applied for description of the dynamic parts of system, we therefore use it for overall design representation; and the latter is adapted for components detail depiction.

4.1 Language Design

In this section, first we discuss the reason why we apply XML for Translation Expression. After a comprehensive reasoning, we adopt XML schema to depict details of structure of Translation Encoding expression.

4.1.1 Reasoning of Translation Expression Description Language

As we discussed in 3.2.3, translation feature description for Provider-Translation scenario could not be completed merely in Key-Value Pair (KVP) (2.3.1), we need XML for representing translation features. Furthermore, three facts support the decision of XML as Translation Expression:

1. WFS can support requests in form of XML encoding through HTTP POST method. It guarantees that our Translation Encoding fragment can technically be seamless incorporated into WFS requests.

2. XML has been universally accepted as an internet standard for data transport and information representation. In turn, most programming languages provide APIs for XML processing. Comparing to stream based file processing, XML processing can benefit from a variety of standardized tools for optimizing XML element locating and manipulating e.g. XSLT (Klausen 2006), XQuery.

3. Considering from HTTP protocol itself, which is the internet transport protocol (W3C 2009) for submitting our Translation Encoding request, applying XML for Translation

Encoding is suitable for information transport based on HTTP. XML-based Translation Encoding fragment can be enveloped into HTTP message body. Hence, it is easier for later programmatically processing and enhancing security of information through encrypt processing.

4.1.2 Translation Encoding Expression Design

Before presenting the encoding expression structure, we list a variety of principles for design:

1. **Loose coupling**. Loose coupling is a classic software design notion and comprehensively applied in computer system design (James and Karl 1990). Here we intend to implement the loose coupling in system through loosing constructive encoding elements coupling.

2. **Reusability**. Reusability will be in phase of applying translation rules and translation products. Our encoding language is intended to be able to freely import existing translation rule and efficiently choose an appropriate realized product for execution.

3. **None-conflict.** Our encoding language should be able to work with other encoded language e.g. filter encoding without conflict. Although specifying a unique namespace can avoid possible conflicts while using elements with the same names, we suggest that our language should be also different in syntax. It can make more easily understanding and readable by human.

4. **Fault-tolerance**. In practical application, system may suspend due to unexpected exceptions and errors. Our encoding language should offer a mechanism to supply a friendly and fully-informational warning while an exception happens.

According to the design principles, we enumerate the properties of our encoding language. The necessary elements are proposed as follows:

1. **Specifying involved data schemas**. Encoding Translation should specify the data schemas and notifies them to WFS service. It is represented by a group of optional element: *SchemaType*.

2. **Specifying the message participating translation processing**. Messages are constructed from data schema specified above. Message indicates the message flow of translation function. Two directional properties of message：*In* and *Out* are from the function perspective. For example, an *In* message is to the function, whereas an *out* message is from the function. This property is represented by the required element *Message* with two optional child elements: *In* and *Out*.

3. **Declaring translation operation.** Here we consider operation as an element in Translation Encoding, rather than any practical operation offered by translation function. It implies that consumers can declare operation with any name, and further, they are allowed to bind the declared operation with real translation function. This strategy guarantees the loose couple of translation declaring with translation products. This procedure is indicated with a required element: *Operation*.

4. **Binding translation function to implemented operation.** In this section, consumer needs to bind each declared operation with one specific translation function or more. The translation function can be exposed as a web service interface or a function running in local system. We apply *Binding* as the element name

The involved Translation Encoding language elements and relations among them follow the depiction above and are summarized and represented in UML diagram (Figure 4.1).

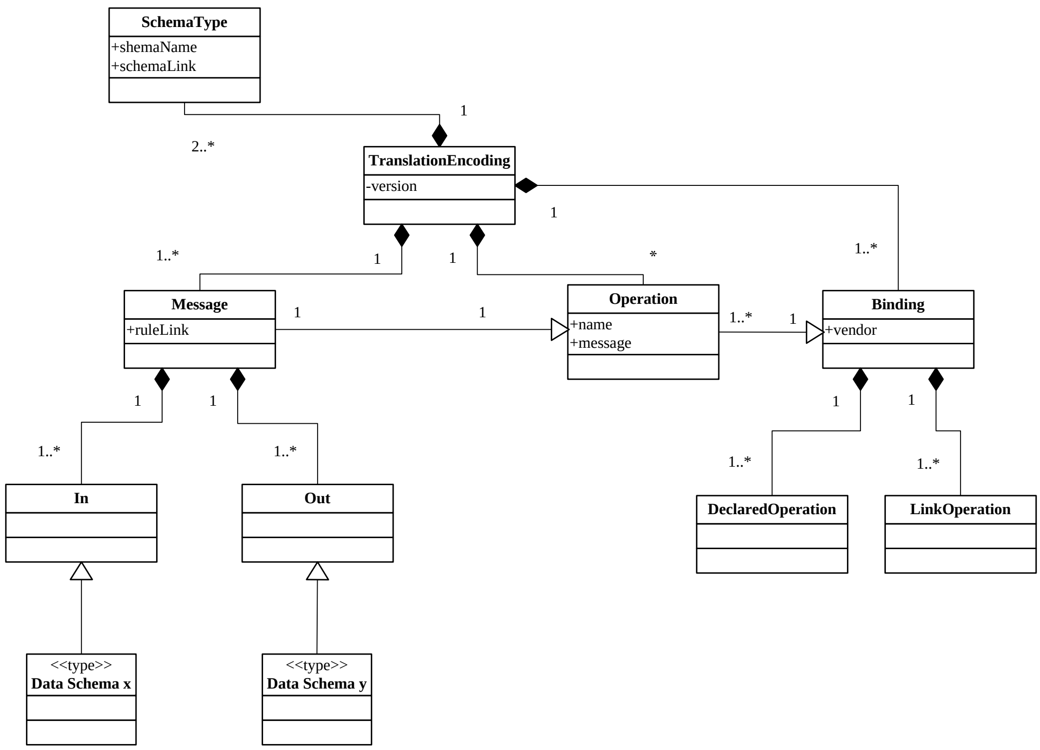


Figure 4.1: UML diagram for Translation Encoding for Provider-Translation.

First, we define a parent element: *TranslationEncoding* for our encoding language. All other elements should be enveloped within it. This element has a set of child elements and one required attribute: *version*, which indicates the version number of our encoding language. Now its default value is ‘1.0’. Based on the discussion of language properties, we develop four child elements of TranslationEncoding, namely: *SchemaType*, *Message*, *Operation* and *Binding*.

SchemaType indicates the data schemas involved in schema translation and it can occur zero or more times within TranslationEncoding. For each SchemaType, it must contain two attributes: *schemaName* and *schemaLink*. The former is a unique identifier for specific data schema, and the later one, whose data type is restricted as validated URI, indicates the link to the schema.

Message must appear once or more. According to whether a consumer submits an existing rule file, there are two manners to set up this element. When a consumer intends to apply a rule file, he or she must specify the attribute *ruleLink* in order to specify the resource link to the rule file. Whereas, the consumer must manually define translation mapping through assigning a couple of child elements: *In* and *Out*. The element In represents the source data schema and Out for target data. All XML schema elements appearing in both elements should have been imported in SchemaType element in advanced. The amount of schema elements enveloped respectively with both indicates the relation of translation mapping. For instance, if one XML schema element is defined in In element, the other in Out element, this translation should be a one-to-one mapping. If more than one declared in Out element, the translation would be a one-to-many mapping. Through combining of SchemaType and Message definition, users can freely define translation mapping relation and also alternatively submit an exiting for translation.

With Operation element, users are able to declare a function interface for translation. The Operation is composed of defined Message element through specifying attribute *message*. One Operation can contain one kind of Message with the translation rule definition. Moreover, it can also pack one group of Messages and hand them over to a specific translation product. The attribute *name* will be the unique identifier for each Operation.

The Binding element connects the declared operation to practical translation functions. Thus the value of child element of Binding: *DeclaredOperation* must be declared in one Operation element already. The other element: *LinkOperation* indicates the URI to a practical translation function. A required attribute: *vendor* must be assigned according to the type of translation implementation. If the implementation is a web service, the value of vendor correspondingly is ‘service’, whereas it would be ‘local’ to a local system. One DeclaredOperation value can be combined with different translation links. If one link is not available, our system will literate other options with this DeclaredOperation value. If failing in obtaining any available links, system will throw a standard exception.

According to the language element analysis above, an XML Schema Document defined for Translation Expression is listed in Listing 4.1.

```
<xsd:element name=" TranslationEncoding " type="TranslationEncoding"/>
<xsd:complexType name=" TranslationEncoding ">
    <xsd:sequence>
        <xsd:element name="SchemaType" type="SchemaType" minOccurs="0"  maxOccurs="unbounded"/>
        <xsd:element name="Message" type="Message" minOccurs="1"  maxOccurs="unbounded"/>
        <xsd:element name="Operation" type="Operation" minOccurs="1"  maxOccurs="unbounded"/>
        <xsd:element name="Binding" type=" Binding" minOccurs="1"  maxOccurs="unbounded"/>
        <xsd:attribute name="version"  type="xsd:string" use="required" />
    </xsd:sequence>
</xsd:complexType>
<xsd:element name=" SchemaType " type=" SchemaType"/>
<xsd:complexType name=" SchemaType ">
       <xsd:attribute name="schemaName"  type="xsd:string" use=" required " maxOccurs="1"/>
       <xsd:attribute name="schemaLink "  type=" xsd:anyURI" use="required" maxOccurs="1"/>
 </xsd:complexType>
<xsd:element name="Message" type="Message"/>
<xsd:complexType name=" Message">
  <xsd:all>
          <xsd:element name="In" type="xsd:string" minOccurs="1" pattern= "NMTOKEN"/>
          <xsd:element name="Out" type="xsd:string" minOccurs="1" pattern= "NMTOKEN"/>
  </xsd:all>
       <xsd:attribute name="ruleLink"  type="xsd:anyURI" use="optional"maxInclusive =”1”/>
    <xsd:attribute name="ruleLink"  type="xsd:anyURI" use="optional"maxInclusive =”1”/>
</xsd:complexType>
<xsd:element name="Operation" type="Operation "/>
<xsd:complexType name="Operation">
      <xsd:attribute name="name"  type="xsd:string" use="required" maxInclusive =”1”/>
      <xsd:attibute name="message"  type="xsd:string" use="required" maxInclusive =”1”/>
</xsd:complexType>
<xsd:element name="Binding" type="Binding "/>
<xsd:complexType name=" Binding">
      <xsd:attribute name="vendor" use="required" default="service"maxOccurs="1">
        <xsd:simpleType>
             <xsd:restriction base="xs:string">
                  <xsd:pattern value="service | local"/>
              </xsd:restriction>
        </xsd:simpleType>
      </xsd:attribute>
      <xsd:all>
          <xsd:element name="DeclaredOperation" type="xsd:string" minOccurs="1"/>
          <xsd:element name="LinkOperation" type="xsd:anyURI " minOccurs="1"/>
      </xsd:all>
 </xsd:complexType>
```

Listing 4.1: XML schema of Translation Encoding for Provider-Translation.

An example about how to specify all elements and attribute is given in Appendix II.

4.2 Function Analysis of System Implementation

Our system will be constructed on client-server architecture. On one hand, through client side user should be able to communicate with an ordinary WFS service and activate the schema translation module. To achieve in this objective, the client must efficiently support for submitting encoded XML fragment through HTTP POST method. In addition, a user-friendly GUI could assist users to succeed in easily communicating with server side. As a consequence, functions for client are summarized as follows:

1. An interface that offers an access to user for submit Translation Encoding fragment enveloped within WFS GetFeature operation. This interface will not only be considered as panel for Translation Encoding editing, but also an implicitly and preparatory syntax validation mechanism to check the Translation Encoding syntax before submit it to server.

2. A GUI that facilitates the communication between client side and server side.

3. An exception notification mechanism to help user check possible faults happened in real time.

On the other hand, server side takes more responsibilities. A component preprocessing translation is mandatory. In addition, in order to offer source data, a software implementation of standardized Basic WFS (2.2.2) is required and it also must be able to handle additional schema translation requests. First of all, server needs to analyze the standard GetFeature operation and extract the content fragment about Translation Encoding. Secondly, it has to understand the Translation Encoding description, and able to select appropriate schema translation products or services for translation execution according to instructions specified by users. In addition, in order to support submission of customized translation rules, server must have the capability of accessing external links to rules and holding memory caching for local duplication of rules. All the function related to schema translation should be encapsulated into an independent and flexible layer of architecture: *Translation Encoding complier*.

Notably, in order to guarantee the flexibility and scalability of system, we suggest a real time binding mechanism for activating translation function. It means that users can communicate with WFS server without concerning about translation function configuration, and only activate translation function automatically by system in real time. Thus, the complier component needs an implicitly way to activate this function. As same as client, server side also needs a fault-handling mechanism. Correspondingly, functions required for server side are summarized as follows:

1. A set of schema translation processing components, which could be implemented as local systems or distributed web services.

2. A standard WFS service implementation, which should support Basic-WFS.

3. A Translation Encoding compiler which must be able to extract Translation Encoding fragment, select processing products according to instructions, and support accessing

external links to retrieving translation rules.

4. A mechanism which facilitates the implicit activation of translation function.

The required functions for both sides are summarized in Table 4.1 and through tool-kits analysis in next section, the development tasks for these function could be simplified or reduced.

Table 4.1: Summary of required functions of Client and Server sides.

Client Side	Server Side
An interface for Translation Encoding request with XML-validation	Schema translation components
A user-friendly GUI	A Basic-WFS service implementation
Client fault-notification	A Translation Encoding complier as a bridge between WFS and translation components
	Standard fault-handling for processing translation

4.3 Analysis of Tool-Kits

The implementation of all required functions on client and server sides can be facilitated through directly applying existing solutions. It avoids the duplicate development and resource waste. However, in order to simplify development tasks, an intensive investigation for existing solution is required. We select some candidates which are separately implemented based on different underlying technologies, and then compare their properties and features to determine the best practical solution. This comparison is come up with both in client developing aspect and in server developing aspect.

4.3.1 Client Products Analysis

There are two basic mechanism can be applied for client implementation: web-browser or standalone application. Web-browser mechanism, also known as Browser-Server, is normally used as a suggested architecture for modern web applications. It is easy for function updating and system security. It implies that most system logic is processed by server and client plays a role for rendering results. Two advantages supports for this mechanism: only through a browser product e.g. Firefox or Internet Explorer, users can obtain services without concerning about system updating; moreover, due to running within browser environment, the local system can not collapse by damages from server processing logic. In GIS industry, many open source or commercial products are developed based on this mechanism e.g. Google Maps API (Google 2010), OpenLayer (OpenLayer 2010). Most of them are capsulated into script language tool-kits and with a standard HTML tag e.g. DIV tag achieve in rendering static maps in the browser. However, we need to per syndic check of Translation Encoding and most script languages e.g. JavaScript, which do not support this XML-Validation. We must consider about the other option: standalone application for our client.

A standalone application offers the full supports for accessing local resource, and plug-in development based on standalone application can obtain comprehensive programming API support. It implies that client tools implemented by standalone application can acquire more user interface components for a more user-friendly GUI, also with the capability of handle XML, the objective of XML-validation can efficiently achieved. We compare different GIS client tools which are all implemented as standalone applications. The candidates includes: JUMP (Martin 2003), uDig (LISASoft 2010). Both two are open source projects and are written in pure Java. Moreover, they both success in supporting OpenGIS Consortium spatial object model and provider API giving full programmatic access to all functions. That means we can extend them with a Translation Encoding client. Therefore, the last critical property for determining which to be the best practical is efficiency and convenience for development. The uDig is developed on the Eclipse RCP (Rich Client Platform) (Jeff 2005), which offers a platform for developing customized plug-ins. Through developing the uDig SDK (LISASoft 2010b) Developers can conventionally add customized processing to standard uDig platform. JUMP also allows GUI-based extension through the JUMP Workbench. However, uDig applies Eclipse RCP framework to simplify the plug-in development, it implies that developer can easily finish most of underlying configuration tasks through graphic interfaces and only need to focus on business logic development. Thus, we recommend uDig works as our client implementation solution.

4.3.2 Server Products Analysis

One important objective of Translation Encoding is to offer interoperability of different schema translation products. According to Provider-Translation mode, the translation function is recommended to be capsulated into web services. So for testing the interoperability, we choose one exiting feasible translation product: FME (SafeSoftware 2008) and develop the other by ourselves.

In addition, a Basic WFS implementation is mandatory. GeoServer (GeoServer 2009) is an open-source geospatial server written in Java and reference implementation of OGC WFS. However, GeoServer does not provider any mechanism to construct a bridge between WFS and schema translation. To implement the extension of GeoServer is beyond this thesis topic. Regardless of lacking a comprehensive developer guide and an overall representation of system architecture, we still recommend GeoServer as the best implementation for WFS and the platform for extension of Translation Encoding.

Reviewing the function table we present in Table 4.1, some developing tasks could be replaced by existing products. A summary of the relation between required functions and software product is shown in Table 4.2. In next section two kinds of languages is intensively discussed for system implementation.

Table 4.2: Summary of required functions for software implementation.

	Required Function	**Implementation Software**
Client	An interface for Translation Encoding request with XML-validation	uDig GUI for Translation Encoding editing and Java API for XML Processing (JAXP) for XML-validation
	A user-friendly GUI	uDig plug-in
	Client fault-handling	uniformed UI warning message from uDig framework
Server	Schema translation components	FME service and one we developed
	A Basic-WFS service implementation	GeoServer
	A Translation Encoding complier connecting WFS and translation components	Developed in Java and be incorporated into GeoServer architecture
	Fault-handling for processing translation	Invoking OGC fault-handling module developed by GeoServer

4.4 Overall Design

We introduce overall landscape for our system according to the function requirement analysis in 4.2. One kind of UML diagrams (Thomas 2002): *use case* can be applied for representing the dynamic relation in system. An overall system landscape is shown in Figure 4.2.

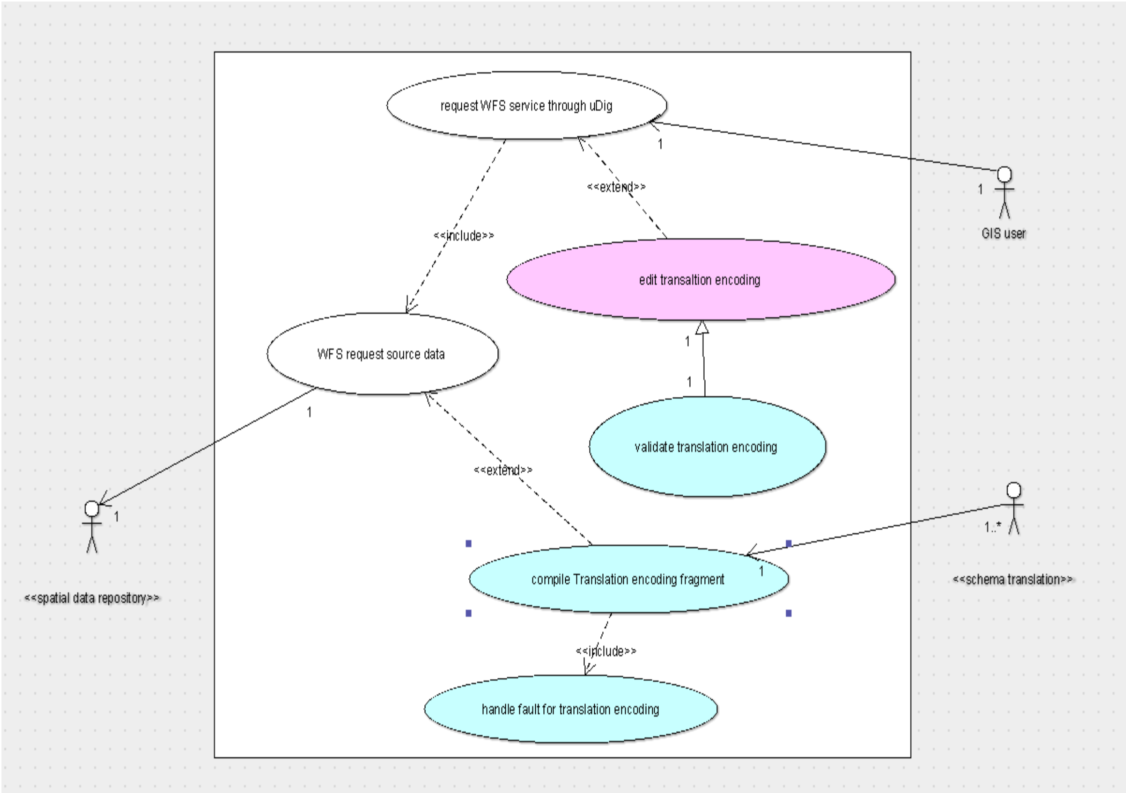


Figure 4.2: Overall landscape of system.

From the Figure 4.2, we consider three actors: GIS user, spatial data repository and schema translation. The latter two are considered as external systems. The schema translation can be any existing translation services or products, e.g. FME. Spatial data repository is the container for spatial data storage. It could be a database product with a spatial data engine, e.g. PostgreSQL (PostgreSQL 2010) with PostGIS (Regina 2009) or a spatial file system e.g. shapefile.

For our system, GIS user directly requests encoded data from WFS service through uDig, and GeoServer as the WFS implementation requesting source data is the mandatory following use case. Editing Translation Encoding and attach it to GetFeature operation is also based on uDig, but it will not involved in every GetFeature requests. An XML-validation check must be work with editing Translation Encoding. The processing of compiling Translation Encoding only happens while the requests from uDig includes translation fragment, and this use case communicate with actor: schema translation. An accessorial use case: handle fault must be incorporated into complier.

Two extend relations guarantee the flexibility of schema translation processing. Notably GIS user can not directly communicate with schema translation. In an ideal situation, users should not be conscious of which concrete product will process schema translation, instead, they only concern about possible participants and compiler can bind a concrete one in real time.

The pinkly marking use cases means that development of those function can be facilitated by framework, and light green marking ones refers to independent development.

After an overall representation of our system, we will focus on component design. In the next section, four components color marked in the Figure 4.1 will be presented in detail with class diagram.

4.5 Component Design

In this section we present primary components of system in details. The UML class diagram is applied for design representation. The components are listed as follows:

1. An interface for editing Translation Encoding.
2. Translation Encoding Validation.
3. Translation Encoding Compiler.
4. One package: modes, representing data models with programming tractable objects.

4.5.1 Interface for Editing Translation Encoding

Although uDig offers a set of plug-in development tool-kits, we intend to avoid to details of developments here and only concentrate on necessary components for Translation Encoding editor. Four main components should be involved for translation editing purpose: a text area for Translation Encoding, a button for merging translation into WFS requests, a button loading up existing translation files and a button for saving text under editing into local file. They all are organized into a container GUI offered by uDig. A menu item added to uDig desktop environment will start this GUI panel. The related UML class diagram is shown in Figure 4.3.

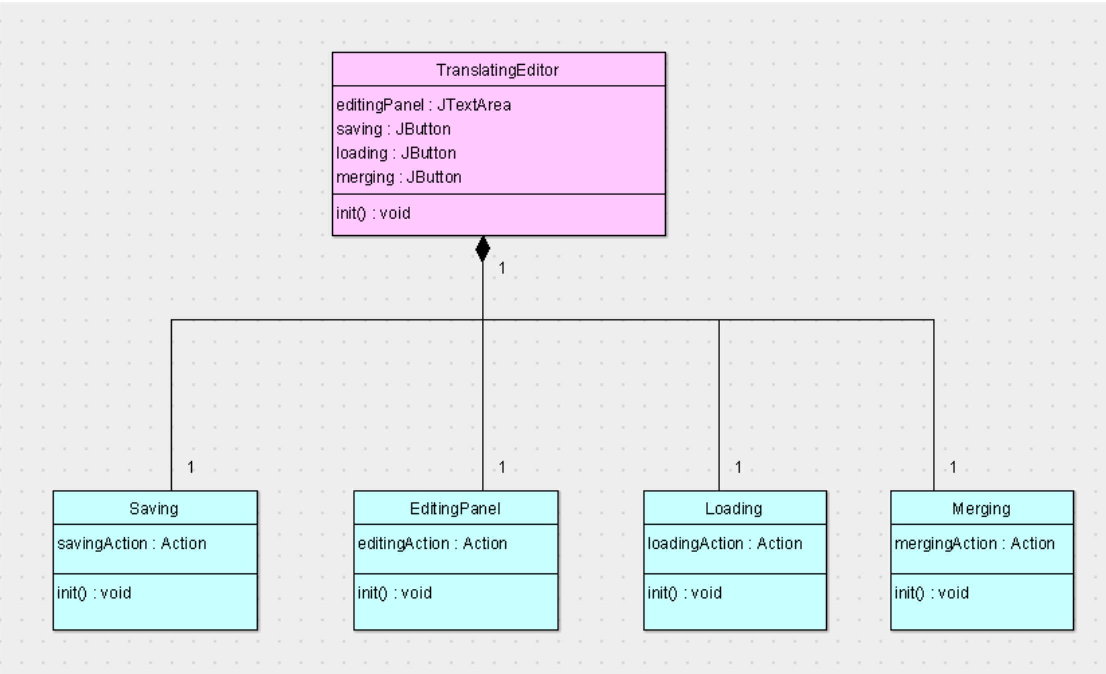


Figure 4.3: Class Diagram of Translation Encoding Editor.

We define four actions for four different GUI components: *savingAction*, *editingAction*, *loadingAction* and *mergingAction*. The behavior and movement vary in each action implementation, and while starting the system up, the action will be automatically loaded into respective GUI component through a method named: init. *TranslationEditor* is a container panel for setting up the layout of these four components, and also it is responsible for bridging uDig plug-in component.

4.5.2 Translation Encoding Validation Component Design

To finish this component: *TranslationValidator*, we use capability for XML validation offered by Java API for XML Processing (JAXP) (Brett 2001), a set of overloading methods provide a variable choices for translation validation, the return value, which is represented as a Boolean data type, is able to notify system whether the user made translation processing fragment is validate. This component should be integrated with saving and merging actions defined in 4.4.1 as a pre-check processing. The related class diagram is shown in Figure 4.4.

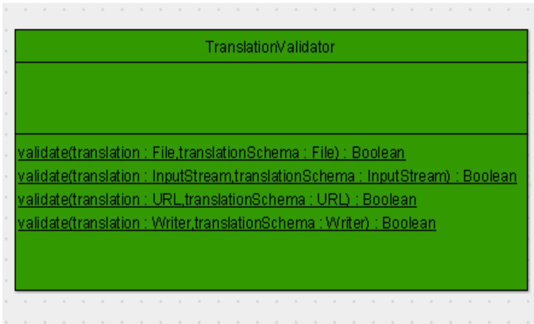


Figure 4.4: Class Diagram of Translation Encoding Validation.

A group of overloading static methods shares the same return type: Boolean. Two input parameters respectively refer to Translation Encoding XML fragment and Translation Encoding XML schema (defined in 3.4). Different parameter types support loading files Translation Encoding contents and schema from system files, Input Streams, URL links or Java Writer object. A comprehensive class diagram representation is shown in Figure 4.5. It renders the relation between TranslationValidator and GUI components.

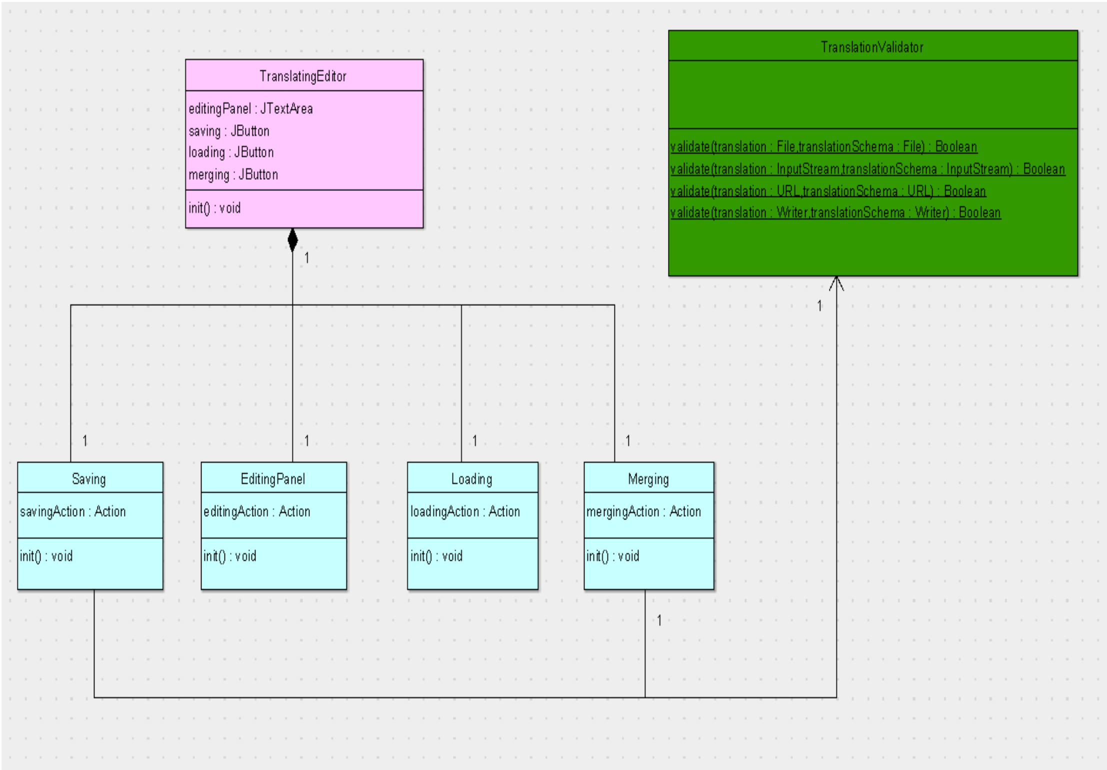


Figure 4.5: Class Diagram of Translation Encoding Validation and GUI.

TranslationValidator (in Figure 4.5) would not directly communicate with TranslationEditor; instead it is interpreted with specific Action class through implicitly invoking it.

4.5.3 Translation Encoding Compiler Component Design

Translation Encoding Compiler plays as the core role in server side. First of all, it must be able to extract translation encoded contents from WFS requests and interpret them to programming objects. In the context of this requirement, we develop a class named: *TranslationCompiler*. The related class diagram is shown in Figure 4.6.

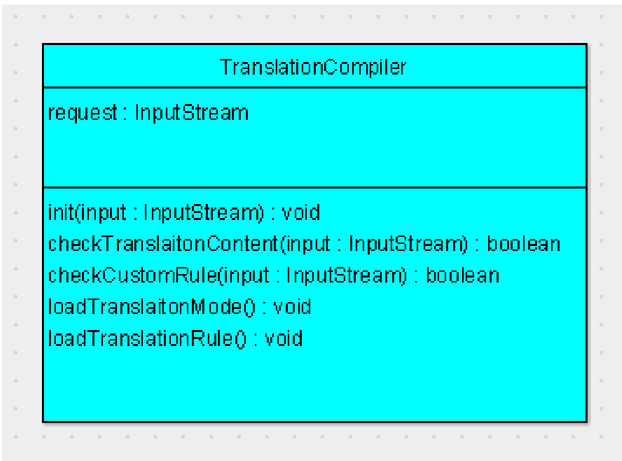


Figure 4.6: Class Diagram of Translation Compiler.

The *init* method will assign WFS requests stream to a variable named request, and then invoke two methods to check whether the request includes translation contents and user-specified translation rules. The method *loadTranslationMode* will interpret searched translation contents to programming modes.

After succeeding in compiling encoded contents, we need to determine whether to build a download service connection to load translation rule depending on users' instructions. If necessary, a class name *TranslationRule* will be loaded through *loadTranslationRule* method and is presented in Figure 4.7

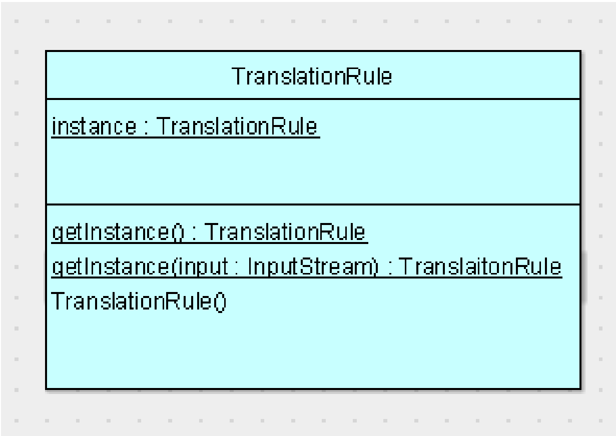


Figure 4.7: the Class Diagram of Translation Rule.

No matter how users intend to communicate with translation services, a connection for accessing external translation services must exist. We name this class *TranslationExecutor*. Notably, before obtaining GML from GetFeature operation and specified translation rule, the TranslationExecutor (shown in Figure 4.8) could not begin with the translation processing. Thus, we intend to implement this class as an observer role in observer pattern (2.3.3) (GOF 1995), correspondingly, TranslationRule and the programming logic in GeoServer for generating GML will be the subjects of the observer, we call them observable. For making sure the loose coupling of system, here we develop a class as parent of TranslationRule and related GeoServer component to make them obtain the observable property. The parent class is named as *TranslationObservable* (Shown in Figure 4.9).

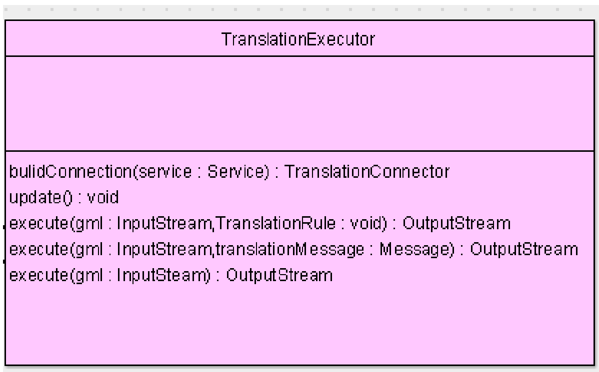


Figure 4.8: Class Diagram of Translation Executor.

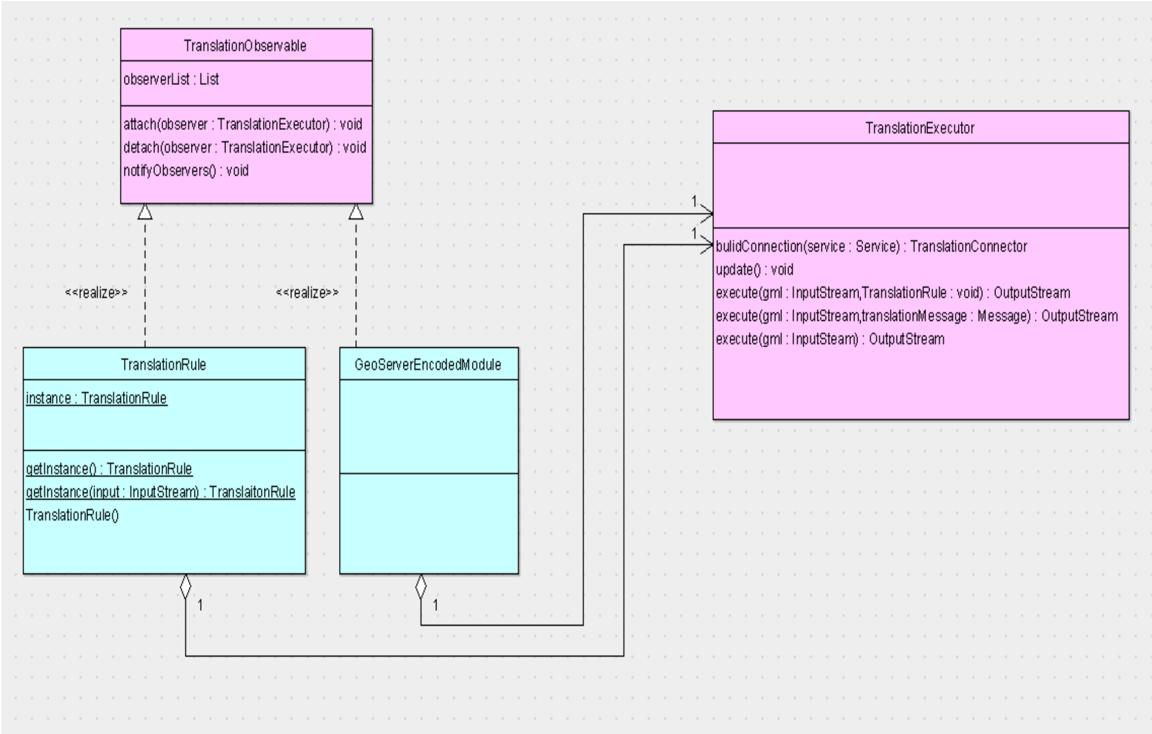


Figure 4.9: Classes Diagram of Translation Observer

Considering the fact we may need several web connections for different purposes, so we can separate this function as an independent module: *TranslationConnector*. The corresponding class diagram and an overall landscape for Translation Encoding compiler component is shown in Figure 4.10.

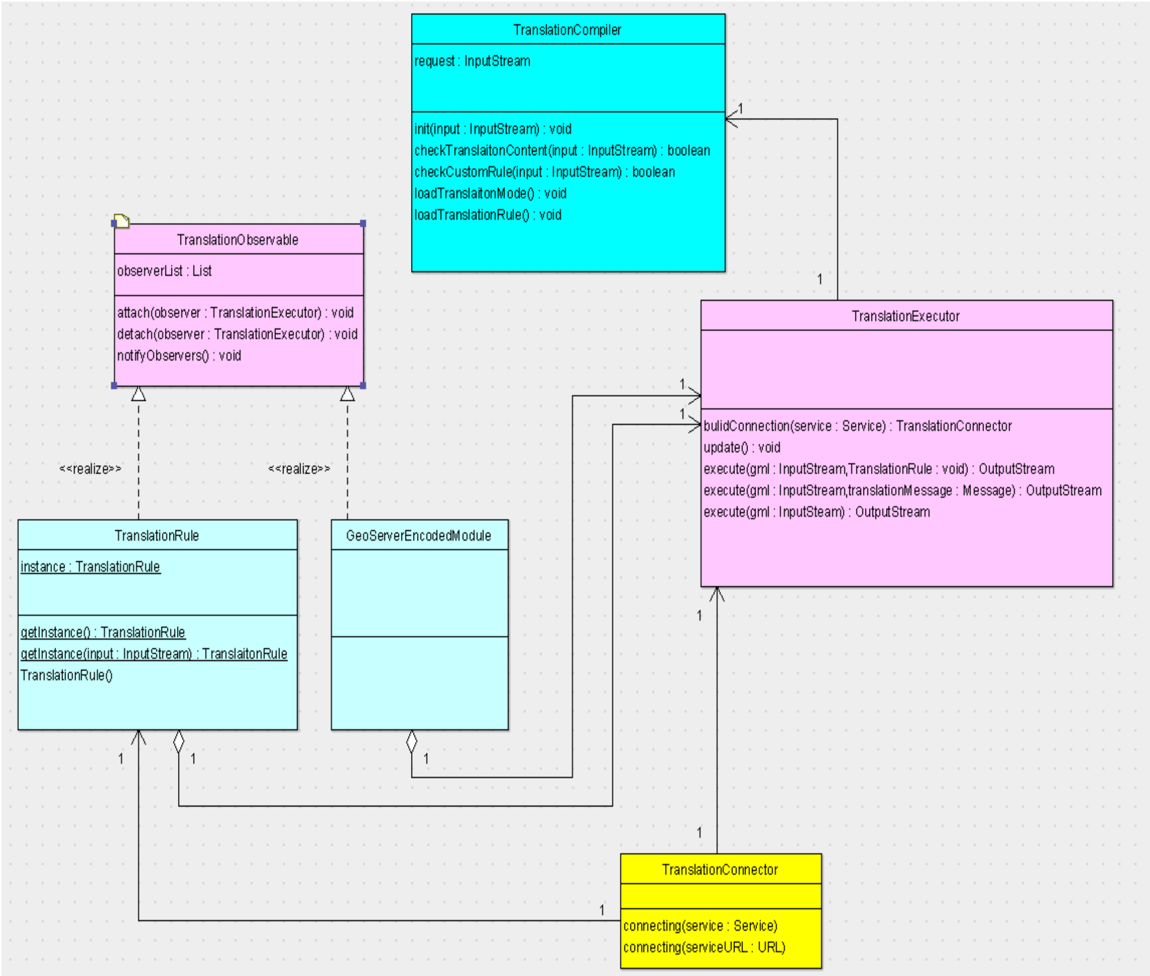


Figure 4.10: Class Diagram of Translation Observer and related components.

4.5.4 Modes of Translation Expression

Data Mode represents the programming objects interpreted from Translation Encoding elements. Thus the class diagram for modes is almost homogeneous to Figure 4.1. However, some operations and attributes are added and we present it here as a mandatory part of the comprehensive components representation (shown in Figure 4.11).

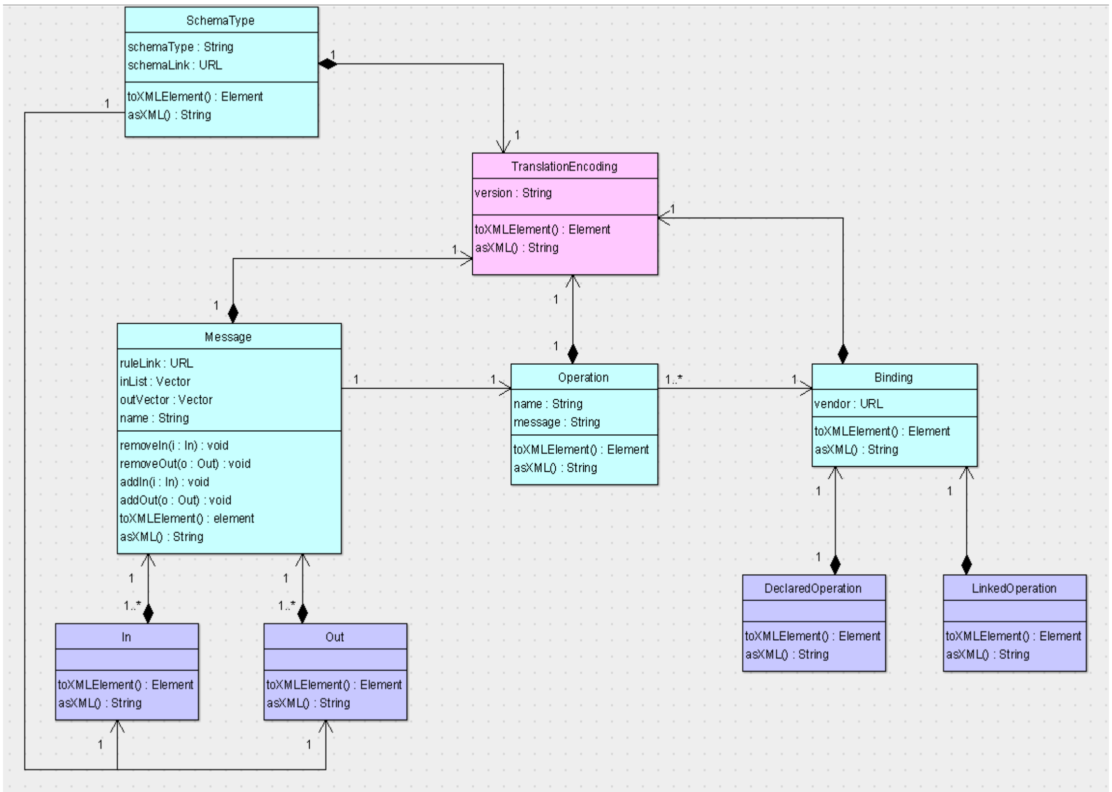


Figure 4.11: Class Diagram of data modes of Translation Encoding.

Comparing to Figure 4.1, this class diagram is aiming at the programming implementation. Each class is attached operations for converting XML file to XML element objects. Moreover, the class *Message* is extended with a set of operation to make it work as container of *In* and *Out* classes. Notably, on one hand, the method *loadTranslationMode* declared in TranslationCompiler is responsible for interpreting XML doc to this mode structure; on the other hand, this mode structure can convert themselves to XML doc.

4.6 Prototype

In this section, we develop a prototype to testing our conceptual design. The prototype is to solve a schema translation task: converting NavLog linear Data into INSPIRE data specification. The translation rule is defined in section 2.1.3. The translation processing is executed by an external translation product we developed by ourselves. The translation is based on two assumptions: the translation rule is stored in a distributed computing platform and published in a web service; the translation product is published in other service.

We develop a web service client which can access GeoServer. It providers a web browser-based GUI for Translation Encoding editing and can send Translation Expression with WFS operations. The GUI is shown in Figure 4.12

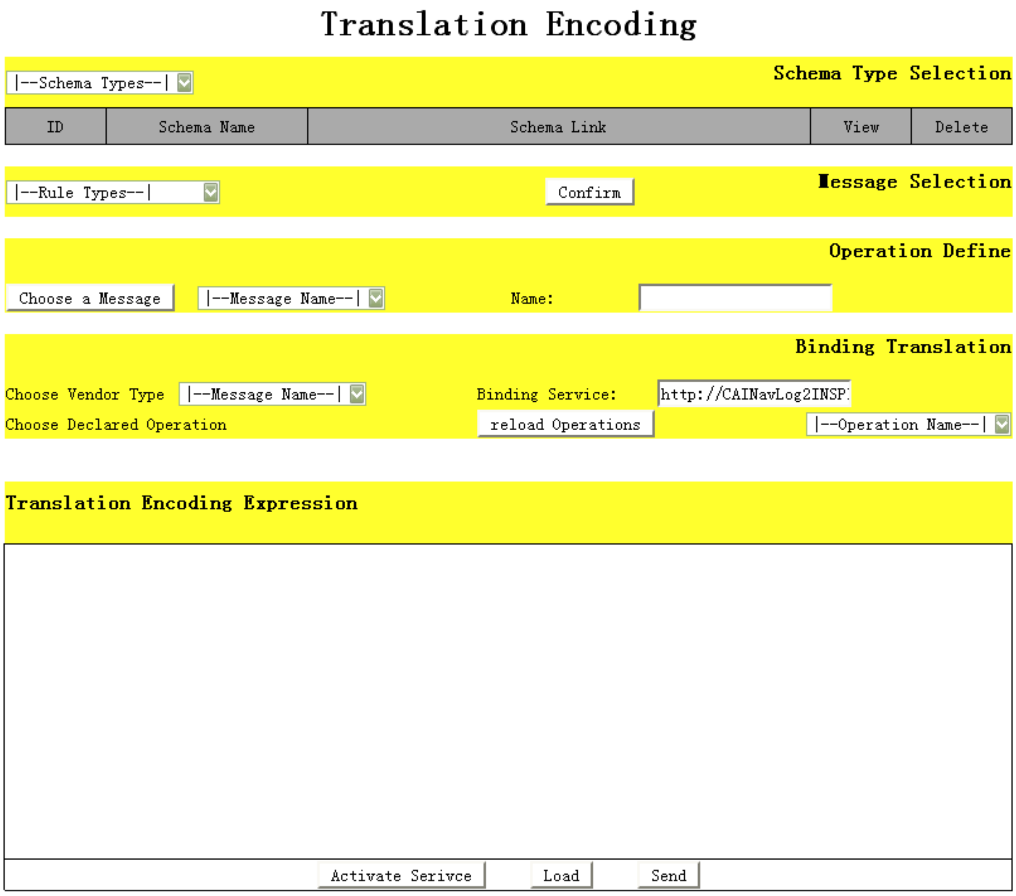


Figure 4.12 GUI for Translation Encoding Client.

According to Translation Encoding language elements, the GUI is composed of four sections: Schema Type Selection, Message selection and Operation Definition and Binding Translation. And a text panel is used for rendering translation expression.

In schema type selection section, users can search schema which needs to be involved in translation (shown in Figure 4.13).

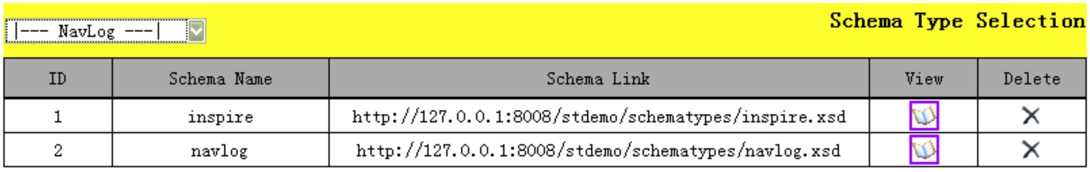


Figure 4.13 GUI for Schema Type Selection.

Users define the message element according to imported schema in this section. For our prototype, we will use existing rule. So we do not specify any schema here.

In order to applying existing translation rule, users need to choose “Remote Rule” (Figure 4.14 (1)) option in Message selection section. According to the specified rule repository, system lists available rule files. The users need to give a name to message and choose a need rule file. Here rule file is stored in http://127.0.0.1:8008/stdemo/rules/ and with a name of

Navlog2INSPIRE.xml (Figure 4.13 (2)).

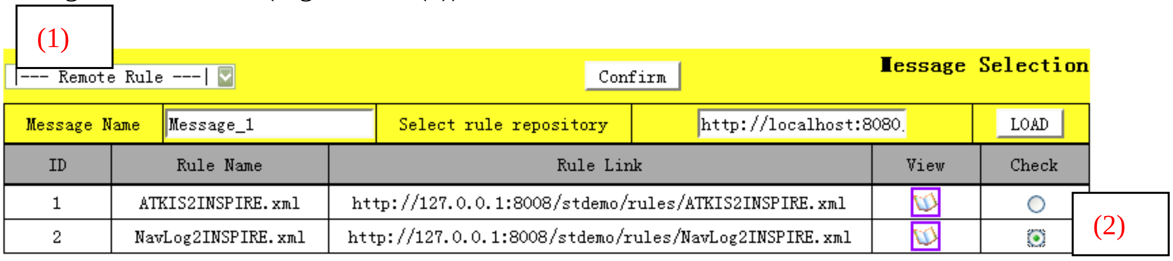


Figure 4.13 GUI for Message Selection.

The defined Message element can be used to specify operation element (shown in Figure 4.14). The message defined in operation must be already specified in Message selection, so Users load available message element through button named “Choose a Message”. Besides that, Users should give a name to the operation. Here the name is “Operation_1”

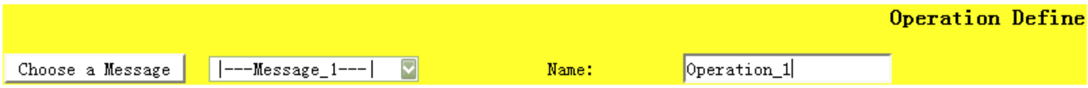


Figure 4.14 GUI for Operation Selection.

After defining operation element, user can specify the last element: Binding (shown in Figure 4.15).

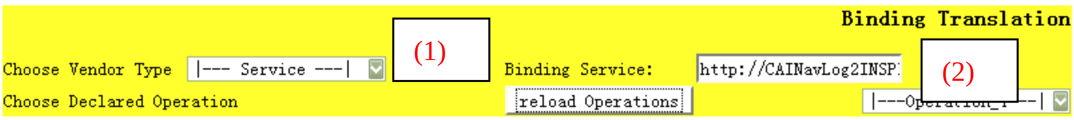


Figure 4.15 GUI for Binding Translation.

Since our translation product is an independent service, here users need to select vendor type as “service” (Figure 4.15 (1)). And users have to specify a web service link for concrete translation product (Figure 4.15 (2)).

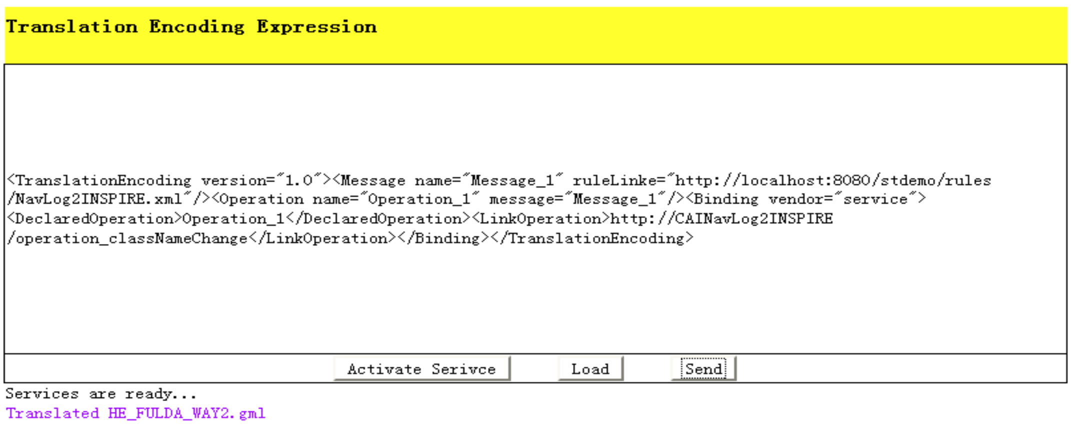


Figure 4.16 GUI for Translation Expression.

Before practical sending translation expression, users can re-check it through a button “load”. The contents of translation expression generates based on the users’ definition in pervious steps. After determining the translation expression, user can send it to with standard WFS request. The translated data is accessible through a hyperlink (Shown in Figure 4.16).

This prototype represents an application which converts data in NavLog linear specification into INPISRE data specification. It achieves a translation processing using external translation products and exiting translation rules. The translation is executed according to the translation expression specify by users. The prototype reveals a few of disadvantages:

1. A web browser-based client may not be appropriate when users need to define customized translation rules.

2. Since we store translation rules and publish translation product respectively in different services. The system is composed of a set of services. The communication among different services may influent system performance.

5. Conclusion and Future work

5.1 Conclusion

To achieve geospatial data harmonization, interoperability of schema translation is considered more critical than implementing schema translation function itself. In this thesis we propose a mechanism to achieve schema translation interoperability in two levels: interoperation amongst existing translation products and interoperation between translation products and OGC services. We compare three manners of integrating schema translation with specific OGC service (WFS) and suggest Provider-Translation as our best practice.

This architecture is based on an assumption that the schema translation processing should be capsulated into an independent web service regardless of implicit techniques of implementation. And a feasible layer called Translation Encoding compiler extends WFS with a capability of schema translation through executing explicit translation services. The compiler first compiles the user instructions called Translation Expression and then chooses the executable translation services either depending on instructions of users, or according to types of translation rule specified by a URL link. Moreover, we deconstruct translation task in a fine grained. It implies that for a schema translation processing, different schema elements can be binding into different translation service in real time. Also we suggest using data schema to represent translation mapping relation.

According to our conceptual presentation, we design a system implementation for our proposed mechanism. We start off by a comprehensive function analysis for system development. And then we compare a variety of exiting GIS products for both client and server sides to facilitate our development task. Some required functions which still need to be developed are presented in detail with UML diagrams. A prototype which can apply external schema translation product through Translation Encoding validates our conceptual design.

5.2 Future Work

Although the Translation Encoding proposed in this thesis can address integration of schema translation with Geo services, it does not have a strong structure to describe schema mapping which can be understood by different products. We propose two possible hypotheses here for future research. One is that we can continue on improving the Message element specified in Translation Encoding expression, and adopt diversity of the quantity of input and output elements to present mapping relation. But it may cause the complexity of implementation. It implies that translation service needs an additional function to convert this representation into their rule structure. The other one may be based on the directly import mapping representations depending on specific product solutions. This manner avoids modification of translation services, but requires users to have a strong knowledge about different rule structures. Thus improving the encoding language to make it describe translation rule in a universally accepted way is most crucial task for our research work.

Now we only choose one specific OGC service: WFS for research. With different underlying mechanism of different services implementation, the manner of integrating schema translation may vary. A more feasible schema translation application needs to be based on a comprehensive and deep research of features of different service types.

Although we present a detailed design for implementation in chapter four, achieving in interoperability of schema translation can not be without an intensive technical investigation. For instance, GeoServer working as an implementation of WFS may become an option for our research, but lacking of completed tutorials for developers and complex system structure may obstruct the extending schema function on it. Thus, offering a best practice as Translation Encoding implementation needs more intensive research.

REFERENCE

Bishr, Y. (1998). Overcoming the Semantic and Other Barriers to GIS Interoperability. International Journal of Geographical Information Science 12(4): 299-314.

Brett, M. (2001). Java and XML Second Edition. O'Reilly. ISBN 0-596-00197-5

Camp to Camp (2008). http://www.spatialdataintegrator.com/, last date accessed: 11.2009.

Dave Chappell. (2004) Enterprise Service Bus. O'Reilly ISBN 0-596-00675-6

Donaubauer, A., Fichtenberger, A., Schilcher, M., Straub, A. (2006). Model Driven Approach for Accessing Distributed Spatial Data Using Web Services - Demonstrated for Cross-border GIS Applications. XXIII International FIG Congress, Munich, Germany.

ESRI. (2009) Retrieved OCT 2009 from http://www.esri.com/software/arcgis/arcgisserver/index.html.

Fielding, Roy T. and Taylor, Richard N. (2002-05), Principled Design of the Modern Web Architecture", ACM Transactions on Internet Technology (TOIT)

Grady Booch, James Rumbaugh, Ivar Jacobson (2005). The Unified Modeling Language User Guide SECOND EDITION. Addison Wesley Professional. ISBN: 0-321-26797-4

GeoServer (2009). GeoServer User Manual Release 2.0.0, GeoServer.

Google (2010). Google Code. Retrieved Jan 2010 from http://code.google.com/apis/maps/documentation/index.html.

GOF. (1995). Design Patterns: Elements of Reusable Object-Oriented Software ISBN 0-201-63361-2

Jaff, M and Jean, L .(2005) Eclipse Rich Client Platform : Designing, Coding, and Packaging Java™ Applications. Addison-Wesley. ISBN-10: 0-321-33461-2.

James Douglas Orton and Karl E. Weick (1990). Loosely Coupled Systems: A Reconceptualization, Academy of Management Review 15 (2):203-223 1990

Kashyap, V. and Sheth, A. (1998). Semantic Heterogeneity in Global Information Systems: The Role of Metadata, Context and Ontologies. Cooperative Information Systems: Current Trends and Directions. San Diego, CA, Academic Press.

Klausen F.M. (2006). GeoXSLT: GML processing with XSLT and spatial Extensions. Department of informatics. University of Oslo

INSPIRE (2004). INSPIRE scoping paper, http://www.ec-gis.org/inspire/reports /inspire _scoping24mar04.pdf, last date accessed: 11.2009.

INSPIRE (2009). INSPIRE Data Specification Transport Networks – Guidelines (Version 3.0.1). INSPIRE Thematic Working Group on Transport Networks.

INSPIRE (2009b). INSPIRE Draft Implementing Rules for INSPIRE Transformation Services (Version 3.0). INSPIRE Networking Services Drafting Team.

INSPIRE (2009c). INSPIRE Draft Technical Guidance for INSPIRE Coordinate Transformation Services (Version 2.0). INSPIRE Networking Services Drafting Team.

Lehto, L. (2007). Schema translations in a Web service based SDI. 10th AGILE International Conference on Geographic Information Science, The European Information Society: Leading the way with geo-information, Aalborg, Denmark.

Leonard and Sam (2007). Restful Web Services. O'Reilly, Sebastopol, CA.

LISASoft. (2010) uDig Installation and Introduction. LISA Soft, Refractions Research.
LISASoft.(2010b) SDK Quick Start. LISA Soft, Refractions Research.
Martin, K. (2009) Java Web Service: Up and Running. O'Reilly ISBN 978-7-121-09711-9
NavLog (2005). Formatbeschreibung zur Erstellung einer forstspezifischen Navigationsdatenbasis - Pragmatisches Shape Forst, http://www.kwf-online.de/deutsch/arbeit/geodat/Spezifikation_pragmatischesshapeForst_3_0.pdf, last date accessed: 11.2009, (German).
Newton, H. (2008). Proposed Requirements Specification for Coalition Information Sharing Center for Advanced Defense Study (2008)
Nicolai, M. (2008). SOA in Practical The Art of Distributed System Design. O'Reilly, Sebastopol, CA.
Nebert, D. (2004). Developing Spatial Data Infrastructures: The SDI Cookbook, Version 2.0. Global Spatial Data Infrastructure.
OGC (2004). OpenGIS® Web Map Service Interface 1.3, Open Geospatial Consortium Inc.
OGC. (2005) Retrieved Oct 2009 from http://www.opengeospatial.org/.
OGC. (2005b). OpenGIS® Web Feature Service Implementation Specification 1.1. Open Geospatial Consortium Inc.
OGC. (2005c). OpenGIS® Filter Encoding Implementation Specification 1.1.0. Open Geospatial Consortium Inc.
OGC (2007). OpenGIS® Geography Markup Language (GML) Encoding Standard - Version 3.2.1. The Open Geospatial Consortium.
OGC (2007b). OpenGIS® Web Processing Service 1.0.0, The Open Geospatial Consortium.
OGC (2007c). OpenGIS® OGC Web Services Common Specification - Version 1.1.0. The Open Geospatial Consortium.
OGC (2007d). OpenGIS® GML 3.1.1 Application schema for Earth Observation products. The Open Geospatial Consortium.
OpenLayer (2010). OpenLayer Users Guide. Retrieved Jan 2010 form http://trac.openlayers.org/wiki/Documentation.
PostgreSQL. (2010) retrieved Jan 2010 from http://www.postgresql.org/
Regina, O. (2009). PostGIS in Action. Manning Publications Co.
Robykiwi. (2010). Retrieved Jan 2010 from http:// en.wikipedia.org/wiki/Ontology_(information_science)
SafeSoftware (2008). Spatial ETL: Making Spatial Data Accessible. White Paper, http://www.safe.com/technology/whitepapers/spatial-etl.php, last date accessed: 11.2009.
Schade, S. (2009). Translation of Geospatial Data: Challenges, Solution and Vision. AGILE Workshop Challenges in Geospatial Data Harmonisation, Hannover, Germany.
Schade, S. (2010). Ontology-Driven Translation of Geospatial Data. AKA, Heidelberg, Germany.
Scott, D.(2009). GIS for Web Developers Adding Where to Your Web Application. The Pragmatic Programmers, LLC.
SnowflakeSoftware (2008). Publishing GML with GO Publisher.
Thomas, A.(2002) UML Weekend Crash Course Wiley Publishing, Inc., Indianapolis, Indiana. ISBN: 0-7645-4910-3.
Martin, D. (2003). JUMP Unified Mapping Platform Technical Report. Vivid Solution

W3C. (2009). drafts -07 of the revised HTTP/1.1 specifications. IETF httpbis Working Group.

W3C. (2004). Retrieved Jan 2010 from http://www.w3.org/XML/Schema#dev

W3schools. (2009). Obtained Jan 2010 from http://www.w3schools.com/schema/schema_intro.asp

APPENDIX

APPENDIX 1: An Example of Translation Encoding for Scenario 1

We assume that a set of geospatial data described in NavLog schema is stored in one spatial database product and published as WFS through GeoServer. And these data are organized as a feature type named: *mydata:muenster*. The URL for this WFS is http://192.168.0.1:8008/geoserver/

And now GeoServer is extended with a schema translation capability which can translation Navlog data into INSPIRE or ATKIS data and vice versa. And there is a user who needs INSPIRE data instead of source schema. So he or she can request a list of supported schema through XML fragment as follows:

```
<GetFeatureModelList    numberOfList='10'/>
```

Or a KVP way like:

```
http://192.168.0.1:8008/geoserver/wfs?service=wfs&
version=1.0.0&
request= GetFeatureModelList&
numberOfList=10
```

The corresponding response should be like:

```
01 <ResponseGetFeatureModelList numberOfList='3' >
02      <Mode modelName='Navlog' schemaLink=' http://192.168.0.1:8008/geoserver/schema/navlog.xsd'/>
03      <Mode modelName='ATKIS schemaLink=' http://192.168.0.1:8008/geoserver/schema/atkis.xsd'/>
04      <Mode modelName='INSPIRE schemaLink=' http://192.168.0.1:8008/geoserver/schema/inspire.xsd'/>
05</ResponseGetFeatureModelList>
```

The attribute numberOfList represents the amount of supported schemas, if it's less than the one specified in request, system will response accurate number regardless of user instruction.

To obtain target data in INSPIRE, the user needs to request GetFeature operation as follows:

```
http://192.168.0.1:8008/geoserver/wfs?service=wfs&
version=1.0.0&
request= GetFeature&
typeName=mydata:muenster
schemaType=INSPIRE
```

Comparing to an ordinary WFS GetFeature request, one parameter called schemaType is given to specify the schema for target data and activate translation processing implicitly.

APPENDIX 2: An Example of Translation Encoding for Scenario 3

We assume that a set of geospatial data described in NavLog schema is stored in one spatial database product and published as WFS through GeoServer. And these data are organized as a feature type named: *mydata:muenster*. An ordinary GetFeature operation should be like this

http://192.168.0.1:8008/geoserver/wfs?service=wfs&

version=1.0.0&

request= GetFeature&

typeName=mydata:muenster

Now one user need translate this data into INSPIRE data schema, and user hold a translation service exposing in link: *http://st.atikstranslator/service/navlog2inspire*, and this translation processing is based on FME. The user also has a FME translation rule file link: *http://www.fme.com/transaltion/rules/navlog2inspire/*.

The user can offer a translation encoding with GetFeature operation as follows:

```
<TranslationEncoding version='1.0'>
    <Message name='myTransMessage' ruleLink=' http://www.fme.com/transaltion/rules/navlog2inspire/' />
    <Operation name=' translatingNavlogToINSPIRE 'message=' myTransMessage'>
    <Binding vendor="service">
        <DeclaredOperation> translatingNavlogToINSPIRE </DeclaredOperation>
        <LinkOperation> http://st.atikstranslator/service/navlog2inspire </LinkOperation >
    </Binding>
</ TranslationEncoding >
```

If the user does not have existing rule and he needs to define a translation rule manually and the translation encoding fragment is as follows:

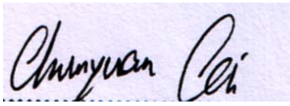
```
<TranslationEncoding version='1.0'>
    <SchemaType schemaName='Navlog' schemaLink=' http://192.168.0.1:8008/geoserver/schema/navlog.xsd'/>
    <SchemaType schemaName='Navlog' schemaLink=' http://192.168.0.1:8008/geoserver/schema/inspire.xsd'/>
    <Message name='myTransMessage1'>      <!-- a one-to-one mapping-->
        <In> <navlog:GEOMETRY> </In>
        <Out> <inspire:centerlineGeometry></Out>
    </Message>
    <Message name='myTransMessage2'>    <!-- a possible many-to-one mapping-->
          <In> < navlog :CLEARWEITH></In>
          <In> < navlog :CLEARHEIGHT></In>
        <Out> <inspire:FormOfWay></Out>
    </Message>
    <Operation name='translatingNavlogToINSPIRE' message='myTransMessage1'>
    <Operation name='translatingNavlogToINSPIRE' message='myTransMessage2'>
    <Binding vendor="service">
       <DeclaredOperation> translatingNavlogToINSPIRE </DeclaredOperation>
       <LinkOperation> http://st.atikstranslator/service/navlog2inspire </LinkOperation >
     </Binding>
</ TranslationEncoding >
```

STUDENT DECLARATION

I declare that the submitted work has been completed by me the undersigned and that I have
not
used any other than permitted reference sources or materials nor engaged in any plagiarism.
All
references and other sources used by me have been appropriately acknowledged in the work.
I
further declare that the work has not been submitted for the purpose of academic examination,
either in its original or similar form, anywhere else.

Declared in Münster 02.02.2010

(Date)



……………………..…………

(Signature)