Ingenieurfakultät Bau Geo Umwelt
Lehrstuhl für Geoinformatik

Geospatial Data Modelling and Model-driven Transformation of Geospatial Data based on UML Profiles

Tatjana Carina Natalie Kutzner

Vollständiger Abdruck der von der Ingenieurfakultät Bau Geo Umwelt der Technischen Universität München zur Erlangung des akademischen Grades eines

Doktors der Naturwissenschaften (Dr. rer. nat.)

genehmigten Dissertation.

Vorsitzender:	Univ.-Prof. Dr.-Ing. Uwe Stilla
Prüfer der Dissertation:	1. Univ.-Prof. Dr. rer. nat. Thomas H. Kolbe
	2. Univ.-Prof. Dr. rer. nat. Gunnar Teege, Universität der Bundeswehr München
	3. Prof dr Jantien E. Stoter, Delft University of Technology

Die Dissertation wurde am 4. November 2015 bei der Technischen Universität München eingereicht und durch die Ingenieurfakultät Bau Geo Umwelt am 4. November 2016 angenommen.

Abstract

Geospatial data modelling plays an important role in the geospatial domain, not least in the context of current regional, national and international spatial data infrastructure (SDI) initiatives, where more and more conceptual models are being developed using formal modelling languages. In particular the Unified Modeling Language (UML) is to mention here, which in the geospatial domain is not only popular in research, but also in professional practice and in standardisation. Besides conceptual modelling, the integration of geospatial data is another important task. Especially within the current SDI initiatives the necessity exists to integrate geospatial data based on differing conceptual models into a harmonised data set conforming to one common conceptual model. Provided that the conceptual models are defined using UML, a model-driven transformation approach for integrating the geospatial data can be applied which follows the Model Driven Architecture (MDA) framework.

However, when integrating geospatial data based on different conceptual models, difficulties can arise from the way these models are currently defined within the geospatial domain. Often, the focus is on an informal and graphical representation of the relevant concepts, to the effect that these models contain errors which hinder their successful application in the model-driven transformation approach. Furthermore, the models often adhere to a variety of UML profiles, some of these UML profiles exhibiting deficits which reduce the quality of the models and, thus, also their machine-interpretability. This includes, in particular, UML profiles defined as part of the ISO 191xx series of geographic information standards. For this reason, model-driven transformation of geospatial data is still rarely applied today, in spite of the well-known advantages this approach brings about in general.

The aim of this thesis is to address exactly the problem of the differing and deficient UML profiles and to provide solutions for how to cope with the variety of UML profiles existing in the geospatial domain and with the deficits they exhibit to allow for the creation of high-quality models and the successful integration of geospatial data using the model-driven transformation approach. Starting with a coherent introduction to the most fundamental terms and concepts related to geospatial data modelling and model-driven transformation which also takes into account relevant standards from the standards organisations ISO, OGC and OMG, the state of the art in modelling and model-driven transformation in academia as well as in professional practice is discussed and predominant problems encountered from the way conceptual models are currently defined and used are illustrated.

Afterwards, a selection of UML profiles currently in use in the geospatial domain is examined. It is discussed, why these UML profiles do not conform to the UML profile definition of the OMG and proposals for how to define these UML profiles in a correct way are provided. Based on these findings, solutions are presented for how to cope with the variety of UML profiles. This includes a generic concept for developing UML profiles in a structured and reusable way, the introduction of a Core UML profile as a universally applicable building block in modelling and model-driven transformation of geospatial data as well as the development of a multi-level information integration framework which allows for transforming between UML models based on differing and deficient UML profiles. Finally, the feasibility and applicability of the Core UML profile and of the multi-level information integration framework is demonstrated by applying them to the transformation of geospatial data from Austria, Germany and Switzerland to the European INSPIRE data specifications.

Zusammenfassung

Die Modellierung von Geodaten spielt im Geoinformationsbereich eine große Rolle, nicht zuletzt im Kontext der gegenwärtigen Geodateninfrastruktur(GDI)-Initiativen, in denen mehr und mehr konzeptuelle Modelle unter Verwendung konzeptueller Modellierungssprachen entwickelt werden. Insbesondere die Unified Modeling Language (UML) ist hier zu nennen, welche im Geoinformationsbereich nicht nur in der Forschung, sondern auch in der professionellen Praxis und in der Standardisierung populär ist. Darüber hinaus ist auch die Integration von Geodaten von Bedeutung. Insbesondere im Rahmen der GDI-Initiativen besteht die Notwendigkeit, Geodaten, die auf unterschiedlichen konzeptuellen Modellen basieren, in einen harmonisierten, zu einem Modell konformen Datensatz zu integrieren. Unter der Voraussetzung, dass die konzeptuellen Modelle mit UML modelliert sind, kann ein auf dem Model Driven Architecture (MDA) Framework basierender Ansatz für die Geodatenintegration angewandt werden.

Sind jedoch Geodaten zu integrieren, die auf verschiedenen konzeptuellen Modellen basieren, so können sich aus der Art und Weise, wie diese Modelle derzeit definiert sind, Schwierigkeiten ergeben. Oftmals liegt der Fokus auf einer informellen und graphischen Darstellung der relevanten Konzepte, so dass diese Modelle Fehler enthalten, die ihre erfolgreiche Anwendung im modellbasierten Transformationsansatz behindern. Des Weiteren verwenden die Modelle oft UML-Profile, welche Defizite enthalten können, die die Qualität und die Maschineninterpretierbarkeit der Modelle reduzieren. Hierzu zählen insbesondere auch UML-Profile, die im Rahmen der ISO-191xx-Normenserie definiert wurden. Aus diesem Grund wird die modellbasierte Transformation von Geodaten, trotz der wohlbekannten Vorteile dieses Ansatzes, bis heute nur selten eingesetzt.

Ziel dieser Arbeit ist es, genau dieses Problem der unterschiedlichen und fehlerhalten UML-Profile zu adressieren und Lösungen aufzuzeigen, wie sowohl mit der Vielzahl an UML-Profilen als auch mit den Defiziten, die diese aufweisen, umgegangen werden kann, um die Erstellung von Modellen hoher Qualität und die erfolgreiche modellbasierte Integration von Geodaten zu gewährleisten. Ausgehend von einer Einführung in die fundamentalen Begriffe und Konzepte der Geodatenmodellierung und modellbasierten Transformation, die auch relevante de-jure und de-facto Normen der Standardisierungsorganisationen ISO, OGC und OMG berücksichtigt, werden der Stand der Technik in Wissenschaft und professioneller Praxis diskutiert und vorrangig anzutreffende Probleme bezüglich der derzeitigen Erstellung und Verwendung von konzeptuellen Modellen aufgezeigt.

Anschließend wird eine Auswahl an derzeit im Geoinformationsbereich verwendeten UML-Profilen untersucht. Es wird diskutiert, warum diese UML-Profile nicht konform zur UML-Profildefinition der OMG sind und es werden Vorschläge unterbreitet, wie diese UML-Profile korrekt definiert werden können. Basierend auf diesen Erkenntnissen werden Lösungen präsentiert, wie mit den UML-Profilen umgegangen werden kann. Dies beinhaltet ein generisches Konzept für die Entwicklung von UML-Profilen in strukturierter und wiederverwendbarer Weise, die Einführung eines Kern-UML-Profils als universell einsetzbarer Grundbaustein in Modellierung und modellbasierter Transformation von Geodaten sowie die Entwicklung eines mehrstufigen Frameworks zur Informationsintegration, das die Transformation zwischen UML-Modellen, die auf verschiedenen und fehlerhaften UML-Profilen basieren, erlaubt. Abschließend wird die Machbarkeit und die Anwendbarkeit des Kern-UML-Profils und des mehrstufigen Frameworks anhand der Transformation von Geodaten aus Österreich, Deutschland und der Schweiz in die Europäischen INSPIRE-Datenspezifikationen demonstriert.

Acknowledgements

This thesis represents the outcome of several years of research conducted at the Chair of Geoinformatics of the Technische Universiät München. First, I would like to thank Prof. Dr. Matthäus Schilcher for giving me the opportunity to join the excellent team at the Chair of Geoinformatics and to work on those research projects whose results form the basis of this thesis. Furthermore, I would like to thank Prof. Dr. Thomas H. Kolbe for his interest in my research topic and for agreeing on taking over the supervision of my thesis when he became the new head of the Chair of Geoinformatics in 2012. I also thank him for the opportunity to work on new research topics which complement the research conducted until then in an ideal way, broadened my scope of knowledge and lead to new insights which also became part of the results presented in this thesis. In addition, I also thank Prof dr Jantien Stoter and Prof. Dr. Gunnar Teege for co-supervising my thesis.

My special thanks go to my colleague Dr. Andreas Donaubauer for all the fruitful discussions related to my research and for having been a reliable mentor during all these years. I also thank him for carefully reviewing the complete thesis and for providing valuable feedback. My thanks also go to Claude Eisenhut for the many enlightening discussions as well as to Dr. Andreas Illert for always providing valuable advice.

The research would not have been possible without the sponsors of the research projects, the German Federal Agency for Cartography and Geodesy, the State Agency for Spatial Information and Rural Development Baden-Wuerttemberg, the Bavarian Agency for Digitisation, High-Speed Internet and Surveying, the Austrian Federal Office of Metrology and Surveying and the Swiss Federal Office of Topography, which I would like to thank for their continuous interest in the project outcomes and also for the provision of the geospatial data used in this work. In addition, I thank the Round Table GIS for the project coordination.

As regards the practical part of this work, I would like to thank the Safe Software team, in particular Don Murray, Philipp Svehla and Dean Hintz, for their constant support and the invitation to spend one week at their headquarters in Vancouver to introduce me to the secrets of working with FME factory pipelines. I also thank my former students Daniel Banfi and Mostafa Elfouly for their efforts in supporting me with the development of the graphical UMLT editor.

Last, but not least, my deepest thanks go to my parents for their unlimited support and understanding.

Contents

List of Figures

List of Tables

Listings

1 Introduction

1.1 Motivation and problem statement

Conceptual modelling allows for specifying in a formal and platform-independent way the content and the structure of those objects from the real world which are in the context of a certain application to be represented by geospatial data. One possibility for defining *conceptual models* is the use of specialised modelling languages. In the geospatial domain, such languages are being researched in particular in the field of geographic database modelling since the early 1990s (Lisboa-Filho, Sampaio et al. 2010). Another possibility provide general-purpose modelling languages which, as the name implies, can be used for many different kinds of modelling tasks. One such modelling language is the *Unified Modeling Language* (UML) (OMG 2011c) which in the geospatial domain is not only popular in various research areas, but also in professional practice as well as in standardisation. The international Open Geospatial Consortium (OGC) standard City Geography Markup Language (CityGML) (Gröger et al. 2012), for instance, specifies an application-independent information model for semantic 3D city and landscape models using UML. Also in the context of current regional, national and international spatial data infrastructure (SDI) initiatives such as the Infrastructure for Spatial Information in the European Community (INSPIRE) (European Parliament and Council 2007), data models are being developed using UML.

Besides conceptual modelling, the *integration of geospatial data* from heterogeneous sources is another important task in the geospatial domain. Especially within the current SDI initiatives the necessity exists to integrate geospatial data based on differing conceptual models into a harmonized data set conforming to one common conceptual model. In this case, the geospatial data need to be transformed such that their structures conform to the desired target model. The integration of geospatial data involves the definition of transformations either at the data format (schema) level or at the conceptual schema level, which is also referred to as model-driven transformation of geospatial data, followed by an execution of these transformation definitions on the geospatial data itself. Defining transformations at the conceptual schema level brings about several advantages such as format-independence and reuse of transformation definitions and is, above that, a well-established approach in the *Model Driven Architecture* (MDA) framework specified by the Object Management Group (OMG).

However, when applying MDA to integrate geospatial data based on different conceptual models, difficulties can arise from the way these models are currently defined within the geospatial domain. Often, when UML is used for developing conceptual models, the focus is on an informal and graphical representation of those concepts the geospatial data are to comprise, to the effect that these models contain errors which are irrelevant when employing them for communication purposes only, but which become relevant, when using the same models for controlling run-time systems, as applies to the model-driven transformation of geospatial data. This aspect relates to the notion of *model quality*; (Henderson-Sellers 2011) states that '[t]he quality of conceptual models, especially those that are standardized metamodels, is an important factor in ensuring successful use of contemporary design and development practices in industry worldwide'. He lists several examples of how the quality of

models can be affected, one of them being the misuse of UML stereotypes. UML stereotypes are a construct provided by UML to be able to adapt UML models to specific platforms or domains. UML stereotypes are always defined as part of a specific UML profile.

Such UML profiles are also in use in the geospatial domain to adapt UML models to specific concepts prevalent in geospatial data modelling. However, some of these UML profiles contain stereotypes which exhibit various deficits and which, thus, reduce the quality of those UML models to which these UML profiles are applied. This includes, in particular, UML profiles which are defined as part of the ISO 191xx series of geographic information standards; these standards take a leading position in the modelling and exchange of geospatial data and, therefore, cannot be disregarded. Above that, individual communities within the geospatial domain often are in need of additional concepts which they define on top of the existing UML profiles, with the result that a variety of UML profiles exists, each of them not only covering partially diverging concepts, but also exhibiting quality deficits of its own.

The existence of this variety of UML profiles, together with the quality deficits they exhibit, affects, in addition, the successful execution of the model-driven transformation of geospatial data; the discrepancy in visual and machine-interpretable representation of the UML models, mentioned above, intensifies this effect. For this reason, model-driven transformation of geospatial data is still rarely applied today, in spite of the well-known advantages this approach brings about in general. While the latter problem can be solved by exercising more care when defining UML models, the problem of the differing and deficient UML profiles persists. The aim of this thesis is to address exactly this remaining problem and to provide solutions for how to cope with the variety of existing UML profiles and the deficits they exhibit to allow for creating UML models of high quality and for successfully executing the model-driven transformation of geospatial data.

The results of this thesis are based on a number of research projects, which were carried out at the Chair of Geoinformatics of the Technische Universität München (TUM) and which the author of this thesis was involved in.

- From 2006 to 2011, the research project *Model-driven approach for accessing distributed spatial data using Web Services – demonstrated for cross-border GIS applications (mdWFS)* was conducted by TUM in cooperation with the Swiss Federal Institute of Technology Zurich (ETH Zurich) on behalf of the federal surveying agencies of Germany (BKG) and Switzerland (swisstopo), the author having participated from 2008 on. The mdWFS project aimed at developing an approach for model-driven transformation of geospatial data, embedding this approach in a web-based environment and demonstrating the feasibility of the approach (cf. section 4.3.3, page 67, for more details on this project). The outcomes of this project have been very promising, however, the transformation was only possible when all source and target models were based on the same modelling language, which resulted in a remodelling of all involved UML models in the form of INTERLIS models.
- From 2010 to 2012, the research project *Prototypical transformation of spatial data to INSPIRE in the cross-border Lake Constance region* was carried out by TUM in cooperation with the company AED-SICAD AG on behalf of the state and federal surveying agencies of Baden-Wuerttemberg (LGL BW), Bavaria (LDBV), Austria (BEV) and Switzerland (swisstopo). The project consisted of three parts. The first part focused on a comparative survey of how modelling and model-driven transformation of geospatial data are currently applied in the Lake Constance region in the context of INSPIRE. The survey resulted in the finding that not all UML profiles in use are compliant to the UML specification and in suggestions of how to cope with this finding and the existing variety of UML profiles in the context of model-driven transformation of geospatial data. The second part consisted in a prototypical transformation of geospatial data to INSPIRE using existing

commercial transformation tools, whereas the third part implemented a model-driven transformation of geospatial data taking into account the outcomes from part one (cf. section 7.1, page 139, for more details on this project). The first and the third part contribute substantially to the contents of this thesis.

- In 2010/2011, the ETH Zurich, on behalf of swisstopo, developed transformations from several Swiss data models to the INSPIRE and EuroGeoNames models based on the model-driven transformation approach from the mdWFS project, the author contributing to this project with support regarding the software tools developed in the mdWFS project.
- In 2014/2015, the research project *Transformation of 3D City Models to INSPIRE* was conducted by TUM, on behalf of LDBV, which aimed, on the one hand, at analysing whether the mapping tables provided by the Working Committee of the Surveying Authorities of the States of the Federal Republic of Germany (AdV) allow for a complete and correct transformation of the Bavarian 3D city model to INSPIRE and, on the other hand, at demonstrating the practical feasibility of the transformation.
- Since 2014, the Chair of Geoinformatics participates in the development of the international OGC standard *CityGML version 3.0*, the author contributing, in particular, to the further development of the CityGML UML model.

1.2 Research questions and methodology of research

Based on the above presented problem statement the following research questions can be derived:

1. To which extent do the ISO-based UML profiles currently in use in the geospatial domain conform to the UML profile definition of the OMG and in which way do they need to be improved when they exhibit deficits?
2. How must a formally correct and universally applicable UML profile for geospatial data modelling be designed and which core concepts must it contain?
3. How can the variety of UML profiles existing in the geospatial domain be structured and designed in a modular way?
4. Is it possible to apply the approach of model-driven transformation of geospatial data in spite of the variety of UML profiles in use, in particular, when these UML profiles do not conform to the UML profile definition of the OMG?

To answer these research questions, the following methodological approach is applied, separating the research work into three parts. The *first part* is empirically oriented. This part focuses on the examination of existing, in particular ISO-related UML profiles, identifies aspects which affect the quality of these UML profiles and provides proposals for how to define these UML profiles in a formally correct way.

The *second part* is formally oriented. Based on the findings from part one, this part (1) presents a generic concept for developing UML profiles in a structured and reusable way, which also accepts the variety of UML profiles existing in the geospatial domain, (2) introduces a new UML profile, called *Core UML profile*, as a universally applicable, fundamental building block for geospatial data modelling and model-driven transformation of geospatial data and (3) presents a *multi-level information integration framework* which allows for transforming between UML models based on different, possibly incorrect UML profiles.

The *third part* is again empirically oriented. This part demonstrates the feasibility of the multi-level information integration framework developed in part two, by transforming geospatial base data from

Austria, Germany and Switzerland to INSPIRE. This transformation incorporates mappings which are defined at the metamodel layer between the UML profiles applied by the UML models from Germany and INSPIRE and the Core UML profile.

The aim of this work is to make the results of this thesis applicable not only to research, but also to professional practice. Therefore, the focus of this work will be on reusing – where possible – concepts from those ISO, OGC and OMG standards which are relevant to geospatial data modelling as well as to model-driven transformation of geospatial data.

1.3 Structure of the thesis

Chapter 2 establishes a coherent terminological basis by introducing the most fundamental terms and concepts related to geospatial data modelling, taking into account relevant standards from ISO, OGC and OMG.

Chapter 3 provides the same fundamental introduction, this time, however, related to the model-driven transformation of geospatial data.

Chapter 4 discusses the state of the art in modelling and model-driven transformation of geospatial data in academia as well as in professional practice and illustrates predominant problems encountered by the author from the way conceptual models are currently defined and used in the geospatial domain.

Chapter 5 examines a selection of partially ISO-based UML profiles currently in use in the geospatial domain. This chapter discusses in detail why the UML profiles and the individual concepts defined therein do not conform to the UML profile definition of the OMG and provides proposals for how to define these UML profiles in a correct way.

Chapter 6 presents solutions for how to cope with the findings from the UML profile analysis. This includes a generic concept for developing UML profiles in a structured and reusable way, the introduction of a Core UML profile as a universally applicable building block in modelling and model-driven transformation of geospatial data as well as the development of a multi-level information integration framework which allows for transforming between UML models based on differing and deficient UML profiles.

Chapter 7 demonstrates the feasibility and applicability of the Core UML profile and of the general framework for multi-level information integration by applying it to the transformation of geospatial data from Austria, Germany and Switzerland to INSPIRE.

Chapter 8 summarises the results achieved within this work, outlines the contributions to scientific research as well as to professional practice and provides an outlook on future research topics evolving from the outcomes of this thesis.

2 Fundamentals of geospatial data modelling

In the context of geospatial data modelling, a wide range of terms and concepts is encountered in scientific literature and daily practice. To a large extent these terms and concepts have their origin in computer science, in particular in the disciplines of Databases and Software Engineering. In general, geospatial data modelling conforms to these terms and concepts; however, sometimes individual terms and concepts were adapted or extended to meet specific requirements inherent to geographic information (Egenhofer 1993) – resulting in meanings which differ from their meanings in the disciplines mentioned above. Furthermore, the same terms and concepts sometimes are used based on a different understanding, leading to imprecise and ambiguous descriptions of geospatial data modelling.

The aim of this chapter is to establish a common terminological basis by introducing those terms and concepts which are, on the one hand, fundamental to geospatial data modelling in general, but, on the other hand, also relevant for the explanations and analyses in the subsequent chapters of this thesis. Since the domain of geographic information intensely makes use of standards from standards organisations, such as ISO, OGC, OMG or W3C, the definitions and explanations provided in this chapter take – whenever appropriate – advantage of exactly those standards relevant to geospatial data modelling. Geospatial data modelling, as covered in this thesis, solely refers to object-based modelling, field-based modelling is not considered.

The chapter starts with a general introduction to the terms *model* and *modelling language*, followed by a detailed description of basic terms related to the concepts *data model* and *metamodel*. The chapter concludes with the modelling language *Unified Modeling Language* (UML), focusing on those UML concepts which are essential to the further understanding of this thesis.

2.1 What is a model?

Depending on the area of application models are to be used in, different concepts of the term *model* apply. An architectural model, for instance, denotes a model which represents the design of a building to scale, whereas mathematical models describe real-world phenomena in a mathematical way and are, for example, used for predicting a possible climate change, simulating the behaviour of fluids or computing the statics of a building. In the context of geospatial data modelling, however, the concept of model as used in computer science is of particular interest. In computer science, models usually are linguistic representations in the form of written or spoken text, pictures or drawings. They refer to the object which is to be represented and, thus, form an abstraction of the modelled object (Hesse and Mayr 2008). The application of models in computer science is by no means a new development, in the database domain, for example, models are in use since the mid-1970s.

2.1.1 Characteristics of models

In 1973 the German philosopher Herbert Stachowiak defined in his book *General Model Theory* three fundamental characteristics which distinguish every model (Stachowiak 1973):

1. *Mapping characteristic*: Models are representations, or mappings, of either a natural or an artificial original. Originals can be anything: objects which exist in the real world, but also non-material things such as imaginations, ideas, concepts or symbols. Furthermore, the original can also be a model itself. Examples for originals within the geospatial domain are individual physical entities such as buildings, cadastral parcels and rivers, but also entire infrastructures such as district heating networks, road networks or also the topography of the earth's surface. The relationship between the original and the model can be complex and diverse. The relationship is not naturally given, but is always influenced by the perception of the particular modeller or model user and, thus, is assigned. Models can also be modified gradually, forming model chains in this way (Hesse and Mayr 2008). This, for instance, holds for the Model Driven Architecture (MDA) approach (cf. section 3.1.2, page 29), where software development consists of a number of subsequent model transformations, in this way generating executable code from platform-independent models. Depending on the intended purpose of the model, its cultural background and environment, and its degree of abstraction, each original can be represented by several models; in the same way, each model can be related to several originals. Thus, original and model are related to each other by m : n mappings (Hesse and Mayr 2008).
2. *Reduction characteristic*: A model usually does not capture all attributes of the original it represents, but only those attributes which appear to be relevant to the modeller or the model user. Thus, models often are simplifications of the real world which have been defined for a certain area of application. Information which is not considered essential can either be simplified, grouped or omitted completely to make the model better comprehensible (Devillers and Jeansoulin 2006) or to adapt it to a certain area of application. However, attributes cannot only be omitted, a model can also be complemented by additional attributes which do not exist in the original (Hesse and Mayr 2008).
3. *Pragmatic characteristic*: This characteristic describes the fact that models are not unambiguously assigned to their originals per se, they rather replace the original under certain conditions and with respect to certain questions. Models are created for certain users and for a certain purpose and they fulfil their task within a specific period of time; thus, they are always bound to a certain task, culture and environment.

These characteristics also hold for geospatial data models. Although, for instance, the INSPIRE data specifications and the German AFIS-ALKIS-ATKIS (AAA) reference model describe overlapping concepts of the real world, they focus on different areas of application – the INSPIRE data specifications focus on environmental applications, whereas the emphasis of the AAA reference model is on geotopography and cadastre –, which also means that they exhibit partially non-overlapping information.

2.1.2 The universe of discourse, conceptual models and conceptual schemas

A fundamental concept underlying any model is the so-called *universe of discourse* (UoD). In general, the UoD comprises all objects being discussed in a certain discourse. The concept of the UoD is widely used in philosophy, semantics, logics and also in computer science. The concept was described for the first time in 1847 by the English mathematician and logician Augustus De Morgan in his work *Formal logic: or, The Calculus of Inference, Necessary and Probable* by stating that 'the whole idea under consideration is *the universe*' (De Morgan 1847). The term universe of discourse itself, however, was introduced in 1854 by the English mathematician, philosopher and logician George Boole in his

work *An Investigation of the Laws of Thought*: 'In every discourse, whether of the mind conversing with its own thoughts, or of the individual in his intercourse with others, there is an assumed or expressed limit within which the subjects of its operation are confined. [...] Now, whatever may be the extent of the field within which all the objects of our discourse are found, that field may properly be termed the universe of discourse. Furthermore, this universe of discourse is in the strictest sense the ultimate subject of the discourse.' (Boole 1854)

In geospatial data modelling, the term UoD is used as well. The standard *ISO 19101-1:2014 Geographic information — Reference model — Part 1: Fundamentals* defines the UoD as a 'view of the real or hypothetical world that includes everything of interest' (ISO 2014); it comprises all objects 'that can be associated with locations on or near the surface of the Earth' (Roswell 2012). This view of the world is strongly related to a certain area of application, which means that the UoD only covers those geospatial objects related to a specific area of application. The UoD is always an imaginary model and never put down in writing. When the UoD is written down *informally*, the standard ISO 19101-1 speaks of a *conceptual model*, which is a 'model that defines concepts of a universe of discourse [emphasis omitted]' (ISO 2014) , whereas a *conceptual schema* represents a 'formal description of a conceptual model [emphasis omitted]' (ISO 2014) and, thus, a model which is *formally* written down. The conceptual model is obtained by abstracting and simplifying the UoD; this means that information which is not considered essential for a certain area of application is either simplified, grouped or omitted completely (Devillers and Jeansoulin 2006). Figure 2.1 illustrates this relationship. The UoD is defined as a conceptual model based on a certain conceptual formalism which, in turn, provides the basis for one or several conceptual modelling languages (cf. section 2.2, page 9); these languages can then be used to represent the conceptual model formally as conceptual schema. The term conceptual formalism can be considered synonymous with the term modelling paradigm (cf. section 2.2.2, page 10) and denotes a certain concept which forms the basis of one or several conceptual modelling languages.

The standard ISO 19101-1, furthermore, lists several principles important for conceptual modelling, amongst others a conceptualisation principle which states 'that a conceptual schema should contain only those structural and behavioural aspects that are relevant to the universe of discourse. All aspects of physical external or internal data representation should be excluded. This requires the production of a conceptual schema, which is independent with respect to physical implementation technologies and platforms' (ISO 2014). According to this principle, conceptual schemas comprise only those parts of the real world which are important to and exist in the UoD. In addition, conceptual schemas are not allowed to contain any information regarding their physical implementation; based on this requirement, conceptual schemas correspond to platform-independent models which are prevalent in the Model Driven Architecture (MDA) framework (cf. section 3.1.2, page 29).

The term *conceptual schema* originates from the database domain and was already mentioned in 1975 in a publication of the Standards Planning and Requirements Committee (SPARC) of the American National Standards Institute (ANSI) in which a *three schema approach* for database management systems (DBMS) is presented. Until then, a DBMS consisted of a two schema structure, an external schema which represents the data as seen by the data user and an internal schema which represents the data as stored physically in the database. The paper introduces a third schema, namely the conceptual schema which 'represents the enterprise's view of the structure it is attempting to model in the data base. This view is that which is informally invoked when there is a dispute between the user and the programmer over exactly what was meant by program specifications' (Steel Jr. 1975). Furthermore, according to this paper 'the conceptual schema is a real, tangible item, made most

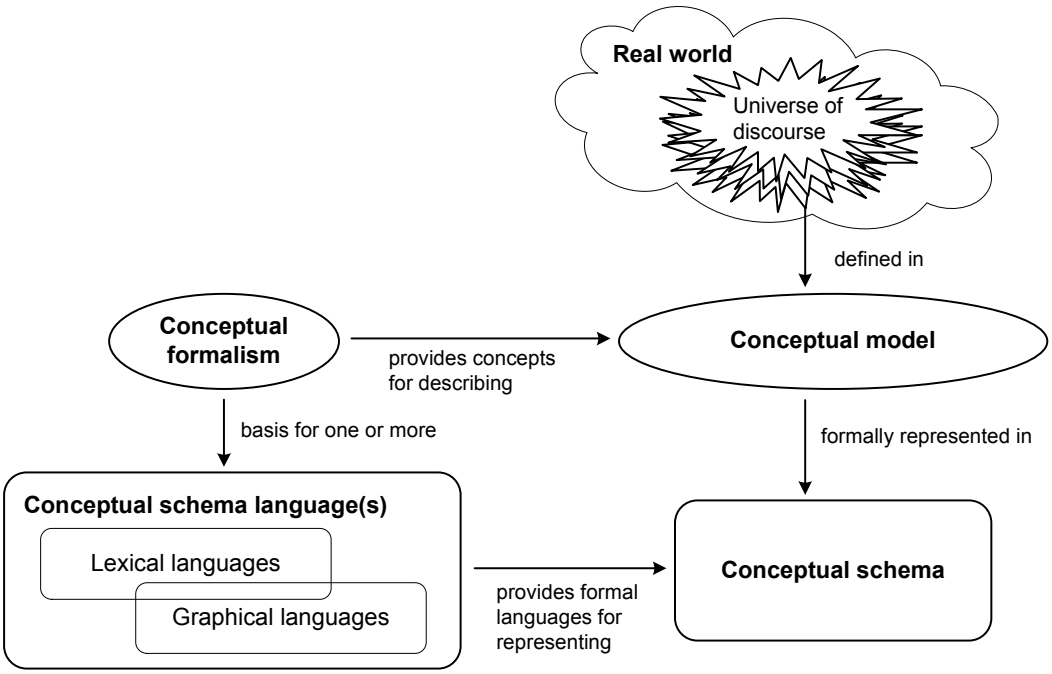


Figure 2.1: From the real world to the conceptual schema (ISO 2014)

explicit in machine readable form, couched in some well defined and potentially standardizable language' (Steel Jr. 1975), which directs us to the topic of modelling languages.

In this thesis the terms conceptual model, formal model or also simply *model* will be used equivalently to the term conceptual schema. On the one hand, the term conceptual model as defined by the standard ISO 19101-1 is of limited significance in the context of this thesis; on the other hand, the term model is the established term used in the computer science domain in general and in the context of UML and model transformation in particular.

2.1.3 Machine-interpretability of models

In computer science, the term *machine-interpretable* generally means that a text, such as a computer program, an XML document or a model, cannot only be read, but also be executed by a computer or by a software tool. The text has to be structured such precisely using a formal language (cf. section 2.2.1, page 9) that its contents and semantics can completely be understood, interpreted and processed by the computer. Neither contradictions nor leeway in decision-making are allowed to exist for the computer. Machine-interpretability does not simply state whether or to which extent a text is *machine-readable*. In contrast to being machine-interpretable, a text which is machine-readable can, in fact, be read by a computer, but the text does not guarantee that its contents and semantics can, in addition, also be interpreted by the computer (Kutzner and Eisenhut 2010).

In the context of this thesis, the term machine-interpretable is used to denote a specific characteristic of models which makes them usable for software development and for controlling run-time systems, as will be explained in more detail in section 4.1, page 53.

2.2 Modelling languages – the medium for describing models

Models can be differentiated into physical models (e. g. architectural, three-dimensional models made of cardboard), linguistic models (e. g. models defined using a modelling language) and mental models (e. g. models which only exist in thought). As already mentioned above, in computer science, linguistic models in the form of written or spoken text, pictures or drawings are prevailing (Hesse and Mayr 2008). To be able to define such a linguistic model a suitable modelling language is required.

2.2.1 Characteristics of modelling languages

In general, languages can be categorised into *formal* and *informal* languages. Formal languages are machine-readable and above that machine-interpretable and they exhibit precise rules; programming languages as well as modelling languages belong to this category. In contrast, natural languages such as English or German belong to the informal languages. Both, formal and informal languages, can exist in *visual* as well as in *textual* form. Among the formal, visual languages is the modelling language Unified Modeling Language (UML) (OMG 2011c), whereas programming languages, such as Java or C++, represent formal, textual languages. Furthermore, also the Extensible Markup Language (XML), which is used for annotating and structuring text, is a formal, textual language, and, thus, also the Geography Markup Language (GML) (ISO 2007), since it is based on XML. Examples for informal, visual languages are pictographic writing systems in use in ancient Egypt and Mesopotamia, whereas nowadays informal, textual languages such as the English language prevail. Each of these four types of languages can be employed for defining models. However, with respect to the definitions given by the standard ISO 19101-1 (cf. section 2.1.2, page 6, and figure 2.1, page 8) informal languages are suited in particular for defining conceptual models which represent the UoD in an informal way, whereas formal languages are preferentially to be used in defining conceptual schemas, since they describe the UoD in a formal way.

Furthermore, every formal and informal language consists of *syntax* and *semantics*. In linguistics, the syntax defines the rules according to which the elements of a natural language (words or symbols) have to be structured to form sentences in a correct way. Similarly, this also holds for formal languages; the syntax of a programming language, for instance, defines how the elements of the language have to be combined to form valid expressions within computer programs. The meaning of sentences, computer programs or models formed in this process is not important, since syntactical rules can be checked without requiring an understanding of the content. The syntax of formal languages can be distinguished further into an *abstract syntax* and a *concrete syntax*. The abstract syntax defines how the individual expressions of the formal language are constructed in general, whereas the concrete syntax defines textual or graphical notation elements for these expressions which enable the programmer to write computer programs using those expressions compliant to the abstract syntax. The abstract syntax of a formal language is often specified in the form of a metamodel (cf. section 2.3.1, page 12); the UML metamodel, for instance, specifies the abstract syntax of the modelling language UML (cf. section 2.4, page 18). The meaning of sentences, in contrast, is defined by the semantics of the corresponding natural language; usually no meaning is assigned to syntactically wrong sentences. Attention has to be paid to the fact that a word or a sentence can exhibit different meanings. The word *bank*, for instance, can denote on the one hand a financial institution where to deposit money and on the other the shore of a river; the correct meaning usually is deducible from the context. Similarly, a meaning is assigned to computer programs and models by the semantics of the corresponding programming language or modelling language, respectively.

This is to be illustrated using the example of the modelling language UML (Hitz et al. 2005): The abstract syntax of UML is defined by the UML metamodel (cf. section 2.3.1, page 12). The UML metamodel determines which syntactical elements exist and in which way they are allowed to be combined to produce well-formed UML models. The concrete syntax of UML, in turn, specifies concrete notation elements for the syntactical elements of the UML metamodel which can be used by the modeller to develop UML models. The semantics, finally, assigns a meaning to the syntactical elements and their concrete notation elements. The concrete syntax and the semantics of UML are defined in textual form and illustrated by examples. The abstract syntax of UML specifies, for instance, the syntactical elements *Class* and *Association* and determines that zero, one or several *Associations* can be assigned to a *Class*. The concrete syntax of UML, in turn, defines that a *Class* is to be represented graphically by a rectangle and an *Association* by a line. The meaning of the concrete symbols, here rectangle and line, however, is defined by the semantics.

2.2.2 Modelling paradigms

Every modelling language exhibits a certain *modelling paradigm*. The decision of which modelling language to use for a certain modelling task, thus, not only depends on the area of its application, but also always on the modelling paradigm underlying a language. The following modelling paradigms are widely-used in the context of geospatial data modelling; they are presented very briefly by highlighting their key concepts and key differences (Kutzner and Eisenhut 2010):

- *Relational paradigm*: This paradigm was introduced by Edgar F. Codd as early as in 1970 and is still in use today in the context of relational databases (Codd 1970). Core of the relational paradigm is the *relation* which stores data. A relation is a table which is described in mathematical form. Each table is identified by a table name and consists of columns, referred to as *attributes*, and rows, the so-called *tuples*. Each attribute defines an attribute name and the type of values the attribute is allowed to take. Each tuple consists of a set of attribute values,i. e. of as many attribute values as the table contains columns. A tuple is also referred to as a *data set*. A relational database in general consists of one or more tables, each table being able to store an infinite number of related data sets.
- *Entity-relationship paradigm* (ER paradigm): Only a few years later, namely in 1976, Peter Chen presented the entity-relationship model (ER model) (Chen 1976). The core concepts of the ER model are the *entity* which is equivalent to a relation in the relational paradigm and the *relationship* which models connections between entities. The ER model is primarily used for defining conceptual schemas which are then implemented as relational databases. The ER paradigm produces static schemas, the dynamic behaviour of the entities and their relationships cannot not represented.
- *Object-oriented paradigm* (OO paradigm): In the early 1990s, the OO paradigm had its breakthrough, focusing on business concepts and their dynamic interaction. The OO paradigm represents a fundamental innovation in programming and data modelling. The focus of the OO paradigm is on the concepts *class* and *object*, classes being used to define entities, whereas objects represent instances of entities. Relationships between classes are referred to as *associations*. A class cannot only define attributes, but also *methods* which allow for modelling the dynamic aspects of software or data. The attribute values can consist of complex and user-defined data types which is especially important in the context of geospatial data modelling, e. g. for modelling geometries. *Object identity*, *inheritance*, *polymorphism* and *data encapsulation* are further important concepts which were introduced by the OO paradigm.
- *Object-relational paradigm* (OR paradigm): This paradigm can mainly be found in the database domain within so-called object-relational databases (Kemper and Eickler 2013). The OR paradigm

combines, as the name already indicates, concepts from the ER paradigm with concepts from the OO paradigm; in particular, it extends the ER paradigm by the OO concepts inheritance, complex and user-defined data types and object identity.

- *XML paradigm*: XML (Extensible Markup Language) is a markup language for describing and structuring information. The core concept of XML is the *element*; elements contain the information in the form of element content, sub-elements and attributes and also describe the information semantically. Since elements can contain further elements as sub-elements, the XML paradigm yields a hierarchical data structure.
- *RDF paradigm*: RDF (Resource Description Framework) is a language for making statements about (web) resources. Each resource is identified by a unique resource identifier (URI). The statements are modelled using RDF models which are based on graphs. The core concepts of an RDF model are the *subject*, which represents the resource, a statement is made about, the *predicate*, which represents an attribute of the subject, and the *object*, which provides the value of the predicate.

2.2.3 Origin and present-day use of the term conceptual schema language

A term often appearing in the context of geospatial data modelling is *Conceptual Schema Language (CSL)*. This term was introduced for the first time in 1979 by B. Breutmann, E. Falkenberg and R. Mauer in their paper *CSL: A Language for Defining Conceptual Schemas* (Breutmann et al. 1979). In this paper, a data definition language was described which can be used for defining conceptual schemas. This data definition language was named CSL, which means that the term CSL can to a certain extent be regarded as a product name for exactly this language only.

However, nowadays the term CSL occurs – in the geospatial domain in general and in the context of the ISO 191xx series of geographic information standards in particular – uncoupled from its original meaning. On the one hand, the term CSL is used in a generic way to denote conceptual modelling languages per se; on the other hand, the standard ISO/TS 19103 uses the term as a synonym for the UML profile defined within this standard (cf. section 5.1, page 76) (Kutzner and Eisenhut 2010).

In this thesis the term modelling language will be used, as this is the established term used in the computer science domain as well and to avoid ambiguities with the CSL defined in the standard ISO/TS 19103.

2.2.4 Domain-specific languages

A *domain-specific language* (DSL) is a formal language which is developed for a certain area of application, or domain, to be able to model those concepts which particularly occur in that domain (Stahl and Völter 2006). Like any other modelling language, also a DSL consists of a metamodel (i. e. an abstract syntax) defining the required domain-specific concepts, as well as of a concrete syntax and of semantics. Examples of well-known DSLs are Prolog and Mathematica and also the database query language SQL.

The opposite of DSLs are *general-purpose languages* which are universally applicable and do not focus on a specific domain. Examples are Java as general-purpose programming language and UML as general-purpose modelling language.

The geospatial domain represents an area of application of its own, thus, the use of DSLs specifically developed for the modelling of geospatial data is conceivable here as well. In practice, up to now mostly the modelling language UML is used. Basically, UML is a general-purpose modelling language, however, the UML specification provides a so-called UML profile mechanism (cf. section 2.4.1,

page 19) which allows for adapting UML to specific domains – turning UML into a DSL in this way (Selic 2007; Selic 2012). SysML (Systems Modeling Language) is an example of such a UML profile which defines a widely-used DSL for the systems engineering domain (OMG 2015).

The geospatial domain, in fact, employs various UML profiles to express concepts inherent to geographic information within UML models. Some of these UML profiles will be discussed in detail in chapter 5, page 75. Examples of DSLs specific to the geospatial domain are the Swiss standard *INTERLIS*, a textual language, which is used in particular in Switzerland for modelling geospatial data (KOGIS 2006) (cf. section 5.5.2, page 109), and *GeoUML*, a textual language as well, which was used in Italy, amongst others, for the development of a national standard specifying the structure of geo-topographical databases for public administrations (Negri and Pelagatti 2014).

2.3 Data models and metamodels

A *data model* denotes a model which describes how data in general – and geospatial data in the context of the geospatial domain – are to be structured. A data model defines which elements of the universe of discourse can exist as physical data instances, for example in the form of a data set in a database, an XML document or Java objects created by a running program. Figure 2.2 illustrates this relationship. Data models correspond to conceptual schemas as defined in the standard ISO 19101-1 (cf. section 2.1.2, page 6).

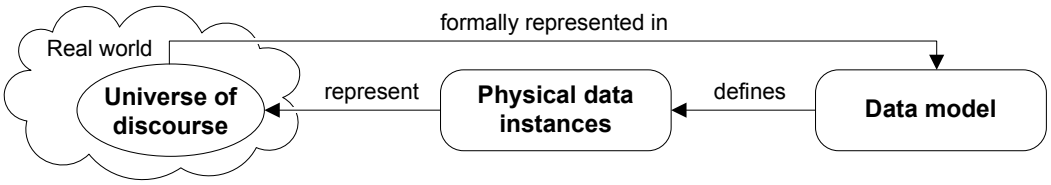


Figure 2.2: Relationship between the universe of discourse, the data model and the physical data instances

Two specific types of data models are prevalent in the geospatial domain, application schemas and implementation schemas; both types will be described in more detail in the following sections. First, however, the concept of the metamodel will be introduced since it is the metamodel which establishes the basis for being able to define data models by means of a modelling language at all.

2.3.1 Metamodels and the four-layer metamodel architecture of UML

A data model is built using a modelling language, the modelling language providing those model elements which can be used to create the data model. For this reason, a definition is required which specifies precisely which model elements are made available through the modelling language. This definition is usually developed in the form of a so-called *metamodel*, a metamodel being a model which defines those model elements that can be used for defining models (Kleppe et al. 2003).

The metamodel concept is to be illustrated by means of the four-layer metamodel architecture used by the UML specification (OMG 2011b)[1]. The individual layers, which are depicted in figure 2.3, have the following meaning (Kleppe et al. 2003):

- *Layer M0*: Layer M0 contains *instances* which represent real-world objects. In the context of geospatial data modelling, these instances are geospatial data representations of real-world objects such as cadastral parcels, buildings or rivers. In figure 2.3 instances of two cadastral parcels and one building are shown together with their properties.
- *Layer M1*: At the layer M1 resides the *model* which describes the structure of the instances at layer M0. The layers M0 and M1 are related to each other since all elements at layer M1 classify and categorise the instances at layer M0 and, equally, each element at layer M0 is an instance of an element at layer M1. Regarding the example in figure 2.3, at this layer the elements *CadastralParcel*, with the properties *label* and *areaValue*, and *Building*, with the property *owner*, are defined of which the elements at layer M0 are instances. In the geospatial domain, the CityGML model (Gröger et al. 2012), the INSPIRE data specifications (JRC 2015) and the AAA reference model (AdV 2009) belong to this layer.
- *Layer M2*: This layer contains the *metamodel* which defines the concepts that can be used to create models at layer M1. The layers M1 and M2 are related likewise, the concepts at layer M2 classify and categorise all instances at layer M1 and each element at layer M1 is an instance of an element at layer M2. At this layer resides, for instance, the UML metamodel which defines the UML language and whose elements can be used to define a UML model such as the INSPIRE data specifications at layer M1. Also UML profiles belong to this layer such as the INSPIRE UML profile (cf. section 5.4, page 100). In the example in figure 2.3, the UML concepts *Class* and *Property* are defined at layer M2 and then used at layer M1 to model the classes *CadastralParcel* and *Building* together with their specific properties.
- *Layer M3*: The elements available in the metamodel must themselves be specified by a model which is called *meta-metamodel* or *metalanguage* and which resides at layer M3. Thus, the meta-metamodel defines all those elements that can be used to create metamodels at layer M2. Likewise, the concepts at layer M3 classify and categorise all instances at layer M2 and each element at layer M2 is an instance of an element at layer M3. The *Meta Object Facility* (MOF) (cf. section 2.4.4, page 25), for example, is the meta-metamodel used for defining the modelling language UML; UML, thus, is an instance of MOF. The example in figure 2.3 uses the MOF concept *Class* to define the UML concepts *Class* and *Property* at layer M2. Another well-known meta-metamodel is *Ecore* which is part of the Eclipse Modeling Framework (Steinberg et al. 2009).

Theoretically, layer after layer could be added now, each layer defining the concepts to be used at the layer beneath. However, the MOF specification defines the uppermost layer to be recursive. This means that MOF not only provides the elements for defining metamodels at layer M2, but that MOF also defines itself; thus, all elements defined at layer M3 are, at the same time, instances of themselves. This recursion is displayed in figure 2.9, page 25.

[1] Please note that according to the OMG MOF 2 specification (cf. section 2.4.4, page 25) metamodel architectures are not fixed to four layers, as was the case with MOF 1. The specification rather allows for defining 'any number of layers greater than or equal to 2' (OMG 2014b). The 'key modeling concepts are Classifier and Instance or Class and Object, and the ability to navigate from an instance to its metaobject (its classifier). This fundamental concept can be used to handle any number of layers' (OMG 2014b). Thus, the use of four layers does not apply by default to all metamodel architectures, but is specific to UML and also some other OMG specifications, such as the OMG Ontology Definition Metamodel and the OMG Common Warehouse Metamodel specifications. Relational database architectures, in contrast, make use of three layers only, namely system table, table and row (OMG 2014b).

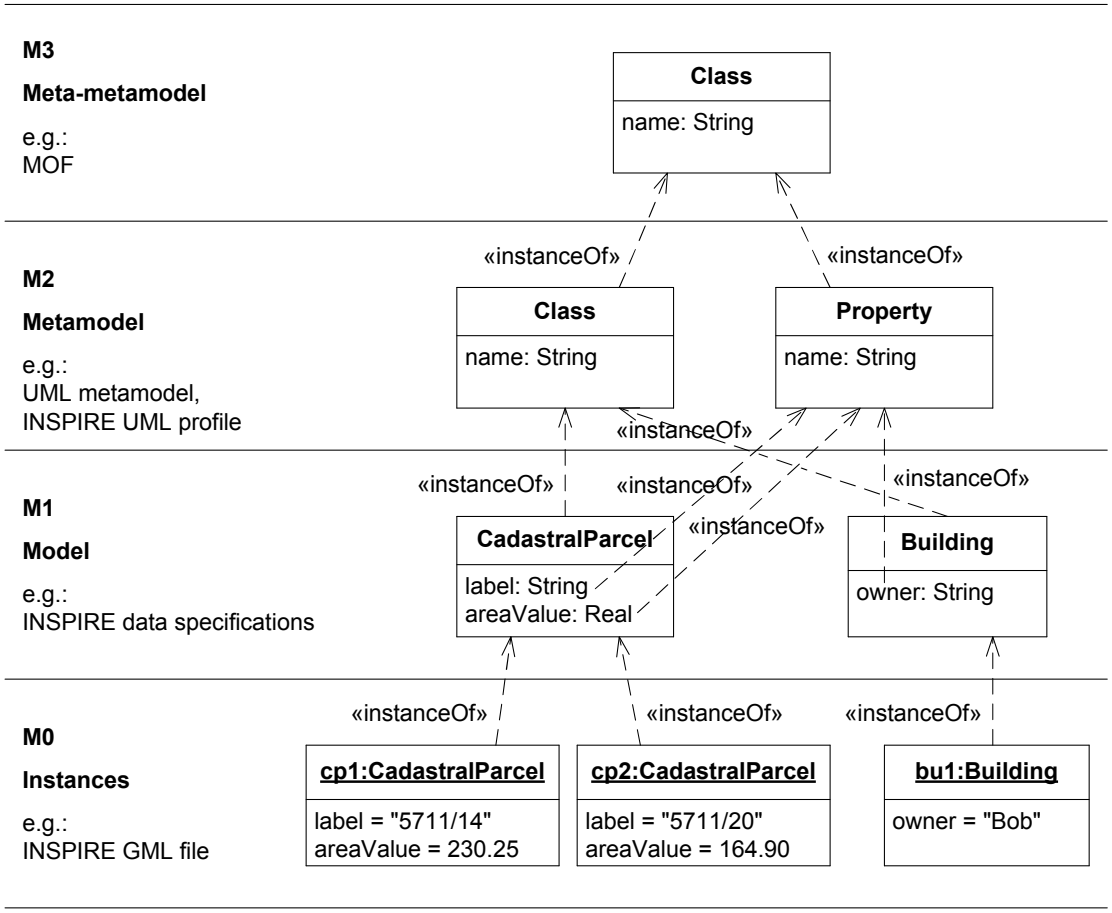


Figure 2.3: The four-layer metamodel architecture of UML (Kleppe et al. 2003, modified)

The four-layer metamodel architecture conforms to a *strict metamodelling* approach which states that in multi-level metamodelling frameworks, the instance-of relationship is the only relationship allowed to exist between models residing at different layers and that it can, in addition, only occur between models residing at adjacent layers. Furthermore, a model A being an instance of model B implies that all elements of model A have to be instances of specific elements of model B (Atkinson and Kühne 2000).

2.3.2 Application schemas and the ISO 19109 General Feature Model

Different areas of application require different geospatial data. An *application schema* is a data model, which describes the semantic structure of geospatial data with respect to a certain area of application the data is to be used in. A corresponding definition is given by the standard ISO 19101-1 which defines an application schema as a 'conceptual schema [...] for data required by one or more applications [emphasis omitted]' (ISO 2014). Thus, application schemas correspond to conceptual schemas and are created using a formal modelling language (cf. section 2.1.2, page 6). Above that, they represent platform-independent models (PIM) as defined by the Model Driven Architecture approach which will be explained in section 3.1.2, page 29. Examples of application schemas in the

geospatial domain are the German AAA reference model, the European INSPIRE data specifications and the CityGML model which have been defined using the modelling language UML.

The standard ISO 19118 relies on application schemas to enable data transfer between two systems and, thus, to 'achieve interoperability between heterogeneous systems' (ISO 2011). The application schema provides for a system-independent semantic and syntactic representation of the data to be transferred which can be interpreted by both, the source and the target system, as is described in more detail in section 3.3.1, page 36.

Based on the ISO 19101-1 definition of application schemas, a *GML application schema*, however, does not represent an application schema in the sense of a conceptual schema, although the name says so. GML application schemas are XML Schema documents, which means that GML application schemas not only define the semantic structure of geospatial data, but in particular also describe the syntax of the corresponding GML instance documents. Often, GML application schemas first are defined platform-independently in the form of UML application schemas and then from these UML application schemas the corresponding GML application schemas are derived according to the *UML-to-GML application schema encoding rules* defined in the standard ISO 19136 Annex E (ISO 2007) (cf. section 5.3, page 86). Figure 2.4 depicts the relationship between UML application schemas, GML application schemas and GML documents.

To be able to define application schemas consistently, the standard *ISO 19109:2005 Geographic information — Rules for application schema* defines rules for how to develop application schemas, facilitating in this way the generation, processing, visualisation and the transfer of geospatial data between different users and systems (ISO 2005a). Basis of each application schema is the so-called *General Feature Model* (GFM). The GFM is a metamodel which was defined in particular for representing geospatial objects, called *features*, their properties and relationships. Figure A.1, page 171, shows an extract from the GFM representing the concepts defined for features. The GFM uses the *object-oriented paradigm*.

The central concept of the GFM is the *feature* which is defined in ISO 19109 as an 'abstraction of real-world phenomena' (ISO 2005a). Such phenomena can be physical entities of the real world, like a building or a street, but also perceived entities, such as an event or an observation. Common to all phenomena in the context of geographic information is that they are 'implicitly or explicitly associated with a location relative to the earth' (ISO 2005a). Features can be characterised in more detail by means of properties, i. e. attributes, associations to other features, and operations. Attributes can be of different types such as a spatial, temporal, locational or a thematic type and they can also represent metadata. Features which are similar and exhibit common characteristics are classified into a *feature type*; the concept of the feature type is represented in the GFM by the metaclass *GF_FeatureType* (cf. figure A.1, page 171). All feature types which are modelled in an application schema according to the GFM have to be instantiated from this metaclass. The concrete features of a feature type are referred to as *feature instances*. For example, the features *Schoenbrunn*, *Nymphenburg* and *Versailles*, which represent abstractions of the corresponding real-world castles in Vienna, Munich and Paris, can be considered as feature instances of a feature type *Castle* which defines characteristics common to all castle-like real-world entities.

The GFM provides a new way of looking at data models in geographic information. The conventional view strongly focuses on geometries, i. e. the object to be modelled is regarded as a point, line or polygon; attributes providing further information on the object are assigned to the geometry representing the object. With the GFM, however, the object to be modelled is regarded as a feature to which attributes are assigned; the geometries of the feature are represented as spatial attributes and are treated in the same way as all the other attributes of the feature. This approach reflects much more the

language of the users of geospatial data, who generally speak of streets, rivers or buildings, but neither of points, lines or polygons nor of tables, lists or tuples (SEE Grid 2010). Another advantage is that a feature can in this way be represented by more than one geometry simultaneously. The feature type *River* can, for instance, have an attribute *river axis* of a line geometry type and, at the same time, an attribute *water area* of a polygon geometry type.

The GFM is a metamodel which provides concepts to be used to create application schemas. With reference to the four-layer metamodel architecture of the modelling language UML (cf. figure 2.3, page 14), the GFM resides at layer M2 and application schemas at layer M1. According to the standard ISO 19109 the 'GFM specifies the requirements for the classification of features, but is not a CSL' (ISO 2005a). This means, although the GFM is represented as UML class diagram (cf. figure A.1, page 171), the standard ISO 19109 only defines the semantics of the GFM, but it does not provide a concrete syntax – in the sense of a syntax of a modelling language – which can be used for modelling application schemas. The GFM could have been represented using any other modelling language as well. Thus, application schemas have to be defined using the syntax of an existing modelling language. The standard ISO 19109 proposes for this purpose the syntax of the modelling language UML, as this facilitates the integration of schemas from the ISO 191xx series of geographic information standards, which have been predefined using UML, into application schemas (ISO 2005a). In addition, the standard provides rules for how to realise the concepts of the GFM in UML to create UML application schemas.

Figure 2.4 depicts the relationship between the GFM and UML application schemas as well as their relationship to the UML metamodel. Furthermore, the relationship to GML is shown. The illustration is embedded into the four-layer metamodel architecture of UML. The figure clearly shows the difference between a UML model and a UML application schema: syntax and semantics of UML models are defined by the UML metamodel, whereas the semantics of UML application schemas is defined by the GFM, but the syntax is taken from the UML metamodel.

2.3.3 Implementation schemas

An *implementation schema* defines the data model based on the concepts of the platform on which the geospatial data described by the data model are to be provided. A similar definition is given in the standard ISO/TS 19103: 'Implementation models [...] describe the realization of domain and system service models in an implementation specific way, related to the characteristics of the underlying platform' (ISO 2005c). This means that implementation schemas are platform-specific and, thus, represent platform-specific models (PSM) according to the Model Driven Architecture approach which will be explained in section 3.1.2, page 29.

Implementation schemas form an intermediate level between platform-independent application schemas, i. e. conceptual schemas as defined by the standard ISO 19101-1 (cf. section 2.1.2, page 6), and platform models (PM), i. e. data (transfer) format schemas. Using implementation schemas, the transition from conceptual schemas to data (transfer) format schemas is conducted in two steps (cf. section 3.3.1, page 36). This approach is of relevance, when a conceptual schema, e. g. a UML application schema, contains constructs which cannot be represented by a specific data (transfer) format schema, e. g. a GML application schema, and, thus, make a direct transformation between the conceptual schema and the data (transfer) format schema difficult.

In Germany this approach is used for deriving the data transfer format schema NAS (Standards-based Data Exchange Interface), which represents a GML application schema, from the AAA reference model, which represents a UML application schema. The AAA reference model has been defined

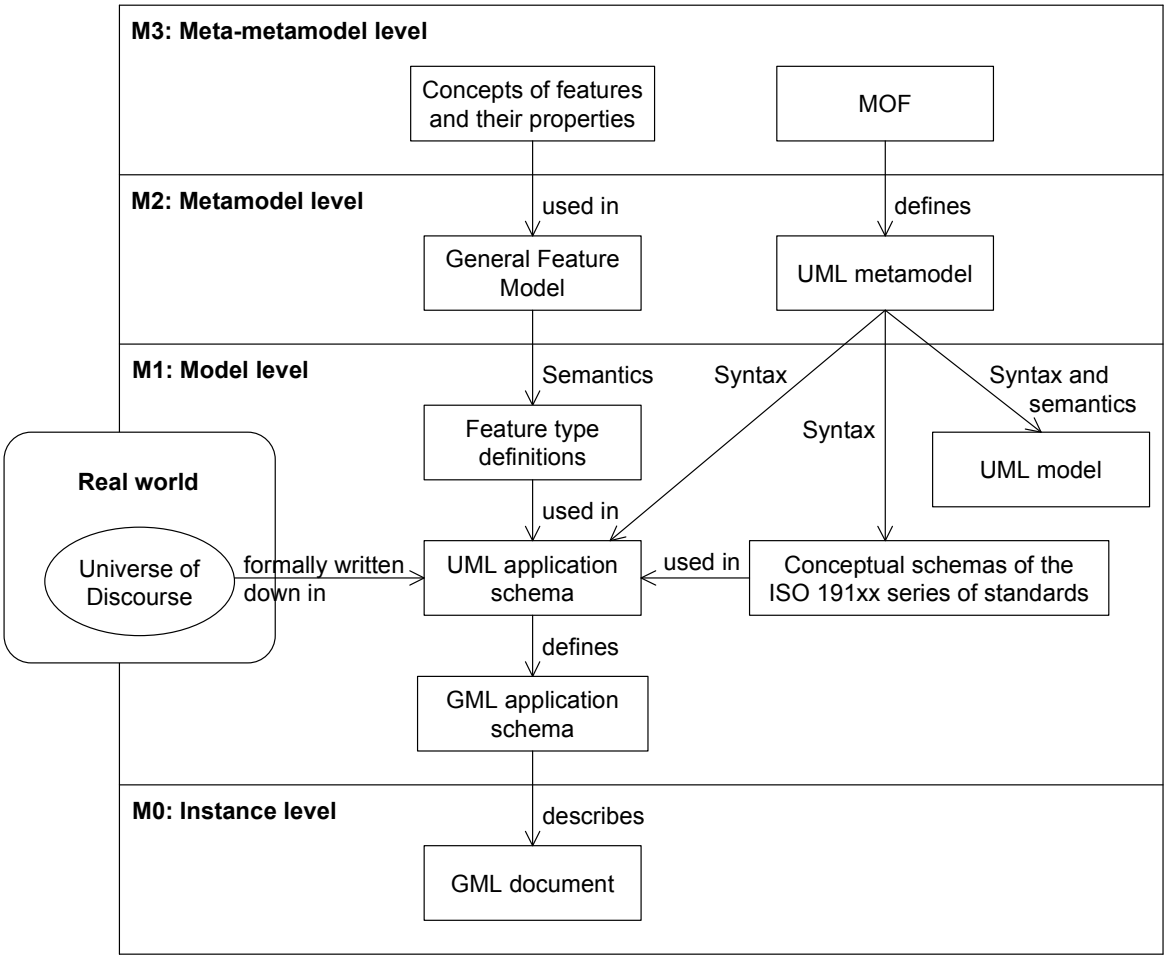


Figure 2.4: Relationship between the General Feature Model, the UML metamodel, UML application schemas and GML (Kutzner and Eisenhut 2010, modified)

using, for instance, the concept of multiple inheritance, which is supported by the object-oriented paradigm, but not by the XML paradigm. Therefore, the UML application schema (PIM), i. e. the AAA reference model, first needs to be transformed into a UML implementation schema (PSM) which conforms to the concepts of the underlying platform NAS; in a second step, the implementation schema is then transformed into the transfer format schema NAS (PM) (Kutzner and Donaubauer 2012; AdV 2014).

2.3.4 Ontologies

Ontologies currently are an important research area in the geospatial domain. According to (Gruber 1995) an ontology can be defined as 'explicit specification of a conceptualization'. An ontology defines vocabulary and concepts for describing and representing a certain area of knowledge. Using ontologies, the world is to be described in a machine-interpretable way such that a machine can process this interpretation. Thus, an ontology represents a model for controlling run-time systems (cf. section 4.1, page 53), which is interpreted by software programs.

Ontologies are, amongst others, to support the exchange of information about certain domains. Several papers discuss the differences and similarities between ontologies and data models. (Spyns et al. 2002), for instance, state that ontologies describe concepts in a generic, application-independent and reusable way, whereas data models focus on specific applications. (Atkinson, Gutheil et al. 2006) conclude in their discussion that ontologies can be considered a subset of models and that most models defined using UML and OCL, in turn, represent ontologies. Similarly, according to (Obrst 2003), ontologies are based on the concepts class, instance, relationship, property, function and process as well as constraint and rule. This is equivalent to the concepts provided by UML and OCL (cf. next section). Thus, UML and OCL provide a means amongst many others, to describe ontologies formally in the form of UML models. For this reason, many UML models, and in particular UML application schemas in the geospatial domain, can be considered to be ontologies as well (Kutzner and Eisenhut 2010).

2.4 The Unified Modeling Language

This section covers the modelling language *Unified Modeling Language* (UML). In the geospatial domain, UML is in particular used for defining data models, also referred to as UML application schemas (cf. section 2.3.2, page 14), in the form of UML class diagrams. It is assumed that the reader is already familiar with the concept of modelling UML class diagrams since, in the following, only concepts relevant for further understanding of this thesis will be explained. This includes, after a short excursion into the history of UML, the *UML profile mechanism* for adapting UML to specific domains, the notion of the *UML keyword* and the *UML package merge* concept.

The modelling language UML (ISO 2012b) is a specification published by the *Object Management Group* (OMG), an international standards consortium which was founded in 1989 and today has several hundred members, mainly companies and organisations from the technology domain. The OMG aims at developing standards for modelling and object-oriented software development. Besides UML, other OMG standards relevant in the context of this thesis are the Meta Object Facility (MOF) and the XML Metadata Interchange (XMI), which will be presented in section 2.4.4, page 25.

The origins of UML date back to the early 1990s when several object-oriented modelling methods appeared, the most important of them having been *OMT* (Object-modeling technique) developed by James Rumbaugh et al., the *Booch method* developed by Grady Booch, and *OOSE* (Object-oriented Software Engineering) developed by Ivar Jacobson. Concepts of these three languages have been incorporated into the development of UML (Hitz et al. 2005). UML 1.1 was the first formal version adopted by the OMG in 1997, followed by several minor revisions (up to UML version 1.5). UML version 1.4.2 has also been formally published by ISO as standard *ISO/IEC 19501:2005 Information technology — Open Distributed Processing — Unified Modeling Language (UML) Version 1.4.2* (ISO 2005b). These early versions of UML are usually referred to as *UML 1.x* or *UML 1*.

At the same time, the OMG started to work on a fundamental revision of the UML standard which was formally published in 2005 as UML 2.0. The latest formal version is UML 2.5 which was adopted by the OMG in 2015. UML version 2.4.1 has also been formally published by ISO as *ISO/IEC 19505-1:2012 Information technology — Object Management Group Unified Modeling Language (OMG UML) — Part 1: Infrastructure* (ISO 2012a) and *ISO/IEC 19505-2:2012 Information technology — Object Management Group Unified Modeling Language (OMG UML) — Part 2: Superstructure* (ISO 2012b). These versions of UML are also commonly referred to as *UML 2.x* or *UML 2*. This thesis is referring to UML version 2.4.1.

While the UML 1 specification consisted of one document, with the release of UML 2 the specification was separated into two documents, the *UML Infrastructure* and the *UML Superstructure*[2]. The UML Infrastructure provides a core for defining metalanguages. This metalanguage core is used in the UML Superstructure to define the UML metamodel, i. e., the abstract syntax of UML which specifies the model elements to be used for creating UML models. The UML Superstructure also defines the concrete syntax of UML, i. e. the notation elements available for representing the model elements within UML diagrams. The metalanguage core of the UML Infrastructure is, moreover, also used for defining other metamodels, such as MOF, the OMG Ontology Definition Metamodel and the OMG Common Warehouse Metamodel. This is illustrated in figure 2.9, page 25.

Within the geospatial domain, UML 1.4.2 and 2.1.2 are those versions which are currently most often referenced by ISO and OGC standards and also by relevant spatial data infrastructure initiatives such as INSPIRE. It is to be assumed that in future revisions of ISO and OGC standards these versions will gradually be replaced by UML version 2.4.1 (cf. sections 5.1.3, page 85, and 5.2, page 85).

2.4.1 The UML profile mechanism

By means of so-called *UML profiles* the UML specification provides a mechanism to adapt the UML metamodel to specific platforms or domains such as the geospatial domain. In this way, UML can be turned into a domain-specific language (cf. section 2.2.4, page 11). The UML specification provides, amongst others, the following definitions:

> A profile must provide mechanisms for specializing a reference metamodel (such as a set of UML packages) in such a way that the specialized semantics do not contradict the semantics of the reference metamodel. That is, profile constraints may typically define well-formedness rules that are more constraining (but consistent with) those specified by the reference metamodel. (ISO 2012b)

> The profiles mechanism is not a first-class extension mechanism (i.e., it does not allow for modifying existing metamodels). Rather, the intention of profiles is to give a straightforward mechanism for adapting an existing metamodel with constructs that are specific to a particular domain, platform, or method. (ISO 2012b)

Thus, the UML profile mechanism only allows UML profiles to extend the UML metamodel in such a way that the concepts defined by the UML profile are still consistent with the concepts specified by the UML metamodel. The UML metamodel itself remains unchanged, it is not modified by the UML profile (ISO 2012b). In this way, a UML model can still be processed by UML tools without difficulties after a UML profile has been applied to this UML model.

In contrast, the UML Infrastructure also provides the possibility to extend the UML metamodel itself, which is referred to as a *first-class extension mechanism*, by introducing, for instance, new associations or metaclasses to the UML metamodel. This approach, however, can result in a modification of the UML metamodel in such a way that UML models are not processable by UML tools anymore.

The core concept of a UML profile is the *stereotype*. By means of stereotypes, UML metamodel elements can be adapted to specific areas of application in order to describe certain concepts of that area of application in a more exact way. Stereotypes can contain properties, which are referred to as

[2] With the release of UML version 2.5 this separation is not being continued; UML version 2.5 is provided as a single document again.

tag definitions, and define *constraints* using, for example, OCL (Object Constraint Language). The tag definitions of a stereotype are equivalent to the properties of a class. The UML profile definition gives the following explanation: 'Just like a class, a stereotype may have properties, which may be referred to as tag definitions. When a stereotype is applied to a model element, the values of the properties may be referred to as tagged values' (ISO 2012b). In geospatial data modelling, however, the tag definitions themselves are usually referred to as tagged values, such as in (ISO 2007) and (JRC 2014a).

Figure 2.5 displays how to define a stereotype. The stereotype is represented using a rectangle. In the upper compartment of the rectangle, the name of the stereotype is written together with the keyword «stereotype», and in the compartment below, the tag definitions of the stereotype are provided. If necessary, the stereotype and the tag definitions can be restricted by constraints. Furthermore, an extension relationship is drawn between the stereotype and the UML metaclass to be extended by the stereotype.
The UML metaclass can, in addition, be marked with the stereotype «metaclass». Moreover, one stereotype can extend several UML metaclasses and one UML metaclass can be extended by several stereotypes. All stereotypes which are defined as part of the same UML profile are grouped into a UML package marked with the keyword «profile» (Hitz et al. 2005).

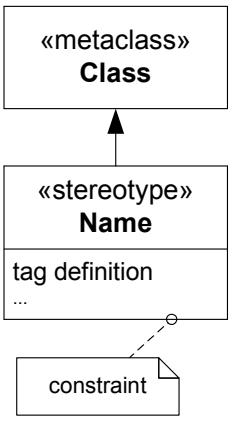


Figure 2.5: Definition of a UML stereotype (Hitz et al. 2005, modified)

To make use of the stereotypes when creating a UML model, the UML profile first has to be applied to that UML package which contains the UML model to be created. To apply a UML profile, a profile application relationship marked with the keyword «apply» is drawn from the UML package to the UML profile. Several UML profiles can be applied to one UML package at the same time, except when they define constraints that conflict each other (ISO 2012b). Furthermore, the UML specification states that when 'multiple applied profiles have stereotypes with the same name, it may be necessary to qualify the name of the stereotype (with the profile name)' (ISO 2012b). Afterwards, the stereotypes can be assigned to the UML model elements either in graphical form via a pictogram or in textual form. When using the textual form, the name of the stereotype is provided between guillemets (« ») before or above the model element to which the stereotype is to be applied. The textual form of applying stereotypes is prevalent in all UML profiles existing currently in the geospatial domain (cf. chapter 5, page 75). It is also possible to apply more than one stereotype to a model element; the stereotype names are then separated by commas. When the applied stereotype contains tag definitions, values can be provided for these tag definitions by attaching the tagged values to the UML model element as UML comment (Hitz et al. 2005).

The distinction of the syntax of UML into an abstract and a concrete syntax also holds for the UML profile mechanism. The definition of stereotypes belongs to the abstract syntax of UML since stereotypes are defined by extending UML metaclasses, which are part of the UML metamodel; the UML metamodel, in turn, defines the abstract syntax of the modelling language UML. The specification that stereotypes are used by assigning their names to UML model elements either in textual form between guillemets or in graphical form via pictograms is determined by the concrete syntax of UML (Noyrit et al. 2010).

An example of a UML profile and its application to a UML package is given in figure 2.6. In this UML profile, called *MyProfile*, the stereotype «FeatureType» is defined as extension of the UML metaclass *Class*. Moreover, the stereotype provides the four tag definitions *documentation*,

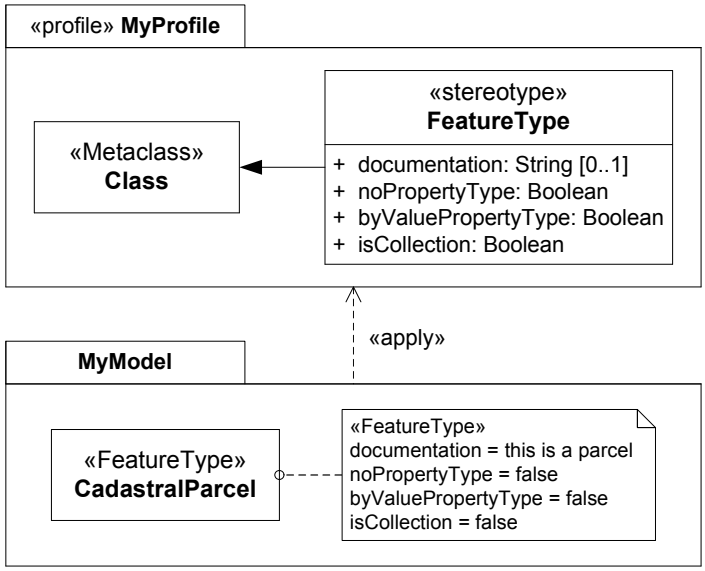


Figure 2.6: Definition and application of a UML profile

noPropertyType, *byValuePropertyType* and *isCollection*. After having defined the UML profile, it is applied to the UML package *MyModel* within which a UML class named *CadastralParcel* is defined. To express that this UML class represents a feature type, the stereotype «FeatureType» is assigned to the UML class and a comment is attached to it providing values for the tag definitions of the stereotype.

The UML specification already provides several standard stereotypes which are defined as part of two UML profiles, the *StandardProfileL2* and the *StandardProfileL3* (ISO 2012b). These stereotypes can be used by any UML model without having to apply the UML profiles beforehand. Regarding the spelling of stereotype names, the UML specification provides the following information: 'Normally a stereotype's name and the name of its applications start with upper-case letters, to follow the convention for naming classes. Domain-specific profiles may use different conventions. Matching between the names of stereotype definitions and applications is case-insensitive, so naming stereotype applications with lower-case letters where the stereotypes are defined using upper-case letters is valid, although stylistically obsolete.' (ISO 2012b) In the context of this thesis, the stereotype names will be used as defined in the corresponding specifications and standards, except for UML profiles which are defined as part of this work; for them always upper-case letters will be used to avoid confusion with UML keywords (cf. section below).

It is possible to integrate existing UML profiles or individual stereotypes into other UML profiles by importing them using the package and element import mechanisms of UML. The stereotypes can then be reused and also be extended through generalisation relationships in the importing profile. This approach, however, can result in complex profiles, in particular, when the imported UML profiles, in turn, reference other UML profiles or stereotypes, which then have to be imported as well to avoid semantic inconsistency. Moreover, complex profiles created in this way can make a separate reuse of the individual UML profiles difficult (ISO 2012b). In contrast, a more practical way of integrating UML profiles is provided by the UML package merge mechanism (cf. section 2.4.3, page 23). When using UML package merge, the individual UML profiles can be merged into a new UML profile which provides all the capabilities of the individual UML profiles in a combined way without the above

mentioned risks the import mechanism can cause. Furthermore, the individual profiles can still be used separately without any problems (ISO 2012b).

The explanations given in this section regarding the definition and application of stereotypes and tag definitions refer exclusively to UML 2. In UML 1, stereotypes are allowed to be defined and applied without having to be associated to a UML profile and also tag definitions can be added to a UML model element irrespective of whether a stereotype has been applied to the UML model element or not. Also, only one stereotype is allowed to be applied to each UML model element.

The UML profile mechanism is not to be confused with the standard *ISO 19106 Geographic Information — Profiles* which provides guidelines for how to define profiles of standards from the ISO 191xx series of geographic information standards (ISO 2004). The profile definition of the standard ISO 19106 is based on a more general definition provided in the standard *ISO/IEC TR 10000-1 Information technology — Framework and taxonomy of International Standardized Profiles — Part 1: General principles and documentation framework* (ISO 1998). Conformance of a profile to the standard ISO 19106 can be established in the following two ways:

> Conformance class 1 is satisfied when a profile is established as a pure subset of the ISO geographic information standards. (ISO 2004)

> Conformance class 2 allows profiles to include extensions within the context permitted in the ISO geographic information standard and permits the profiling of non-ISO geographic information standards as parts of profiles. (ISO 2004)

This means that a profile either represents a restriction (Conformance class 1) of an ISO standard and, thus, only consists of a subset of the constructs provided by the ISO standard; or it represents an extension (Conformance class 2) and, thus, can also contain constructs which do not yet exist in the ISO standard, but may be created in conformance with the extensibility guidelines of that specific ISO standard. Compared with the profile definitions given by the standard ISO 19106, a UML profile would always correspond to Conformance Class 1.

2.4.2 UML keywords

In UML, the guillemet notation is not only used for stereotypes, but also for UML keywords which ‘are reserved words that are an integral part of the UML notation’ (ISO 2012b). Thus, UML keywords represent notation elements which are part of the UML syntax. One of the functions of UML keywords is to distinguish between metamodel elements and between relationships which use the same visual notation. The UML specification defines, for instance, a rectangle as notation for the metamodel element *Classifier*; likewise, all metamodel elements derived from *Classifier* use the rectangle as notation as well. These derived metamodel elements are, amongst others, *Class*, *Enumeration*, *DataType* and *PrimitiveType*, as can be seen in the figures A.2 and A.3, page 172. However, to clarify which metamodel element exactly is referred to when using the rectangle, a corresponding keyword is added to the rectangle. The name of the keyword is defined in the notation section of each metamodel element in the UML specification; for the metamodel elements derived from *Classifier*, the UML specification defines, for instance, that an enumeration is marked with the keyword «enumeration», a data type with the keyword «dataType» and a primitive type with the keyword «primitive». A class, in contrast, is always modelled without keyword, since it is the most common classifier used in UML class diagrams (ISO 2012b).

UML keywords usually start with lower-case letters, whereas stereotype names should preferably start with upper-case letters. UML metamodel elements which have a keyword and which, in addition, are extended by a stereotype are to be represented in UML models such that 'the stereotype name will be displayed close to the keyword, within separate guillemets (example: «interface» «Clock»)' (ISO 2012b).

UML keywords 'have special significance in the context in which they are defined and, therefore, cannot be used to name user-defined model elements where such naming would result in ambiguous interpretation of the model' (ISO 2012b). This means that naming a stereotype equally to a UML keyword is to be handled with care, as it might not be clear, whether the model element represents a stereotyped element or a keyworded element, resulting in different element semantics. This difference is not always taken care of in geospatial data modelling, as will be illustrated in chapter 5, page 75.

2.4.3 The UML package merge concept

Package merge is a concept introduced in UML 2 which allows for combining (i. e. merging) the contents of two UML packages related to each other by a *PackageMerge* relationship (ISO 2012b). Package merge is a directed relationship between a *merged package* and a *receiving package*, as is depicted in figure 2.7, the merged package providing content (merged elements) which is to be merged with the content (receiving elements) of the receiving package; thus, the merged package extends the content of the receiving package. A package merge relationship is denoted with the keyword «merge».

A package merge redefines the merged elements by merging them with the receiving elements according to predefined constraints and transformations. When merging two UML packages, the content of the merged package is added to the namespace of the receiving package. In contrast, a package import makes the elements of the imported package only visible in the namespace of the importing package, but does not automatically redefine them. A redefinition of the imported elements would have to be done manually, for instance, by using generalisation relationships in the importing package (Hitz et al. 2005).

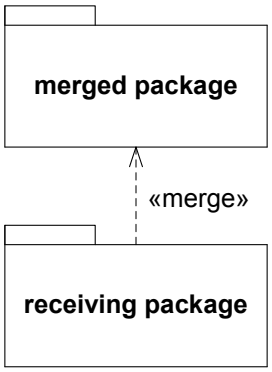


Figure 2.7: The UML package merge concept (Hitz et al. 2005)

The UML specification recommends using the mechanism when 'different definitions of a given concept for different purposes, starting from a common base definition' (ISO 2012b) are to be provided. It is then possible to extend this base definition incrementally 'with each increment defined in a separate merged package' (ISO 2012b). Package merge is especially useful in meta-modelling and was applied extensively in the definition of the UML metamodel.

The semantics of package merge is defined in the UML specification by individual constraints and transformations for the UML metatypes *Package*, *Class*, *DataType*, *Property*, *Association*, *Operation*, *Enumeration* and *Constraint*. Whether a merged element and a receiving element match, and can therefore be merged, depends in general on their name and metatype. Theoretically, the UML package resulting from a package merge represents a set union of the merged package and the receiving package. However, the transformations are carried out implicitly only, which means that the merging solely takes place conceptually, physically the receiving package remains unchanged and, thus, also the physical model in the model repository remains unchanged (ISO 2012b).

This is to be illustrated by the example in figure 2.8 which is based on an example from the UML Superstructure document. On the left side of the figure, a package merge between the merged packages *P1* and *P2* and the receiving package *P3* is represented. The right side of the figure shows how the *resulting package P3* looks conceptually after the merge has been carried out. The resulting package is defined in the UML Superstructure as 'the package that, conceptually, contains the results of the merge. In the model, this is, of course, the same package as the receiving package, but this particular term is used to refer to the package and its contents *after* the merge has been performed' (ISO 2012b). Similarly, the *resulting element* refers to 'a model element in the resulting package *after* the merge was performed' (ISO 2012b). Generally, the receiving package only defines new elements or redefines elements already existing in the merged package. In the example, *P3* is the receiving package; it redefines class *A* by defining two additional properties for the class. The same class *A* is also defined in the merged packages *P1* and *P2*. When merging packages, elements which exist in the merged package and in the receiving package are merged into resulting elements. Elements which are only available in the merged package are simply copied to the resulting package. In the example, the packages *P1*, *P2* and *P3* are merged and, thus, the classes *P1::A*, *P2::A* and *P3::A* are merged into one resulting element, whereas the classes *P1::B* and *P2::C* are copied to the resulting package *P3*, as are the generalisation relationship between *P1::A* and *P1::B* and the association between *P2::A* and *P2::C*.

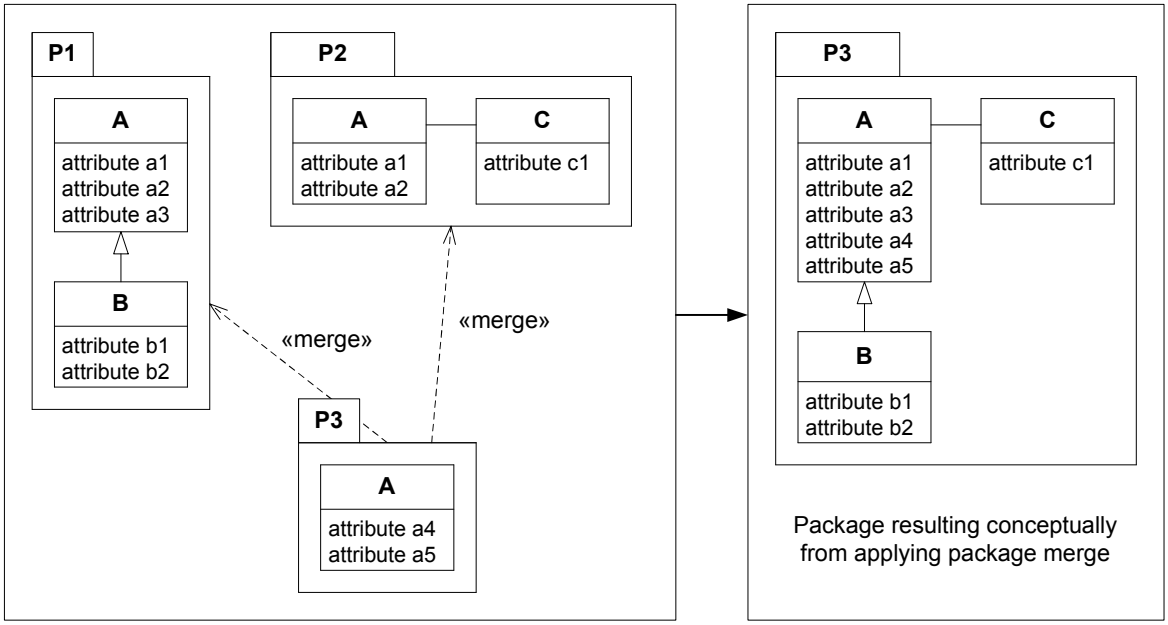


Figure 2.8: A UML package merge example (ISO 2012b, modified)

UML package merge shows certain parallels to the inheritance concept of the OO paradigm. Using inheritance, a UML subclass only defines those features which it does not inherit from the UML superclass; similarly, using UML package merge, the receiving UML package only defines those concepts which it does not merge from the merged UML packages. The difference, however, is that physically the receiving package remains unchanged, whereas instances of the UML subclass will indeed hold the inherited features.

2.4.4 Related OMG standards – The Meta Object Facility, the XML Metadata Interchange and the Object Constraint Language

Three other important OMG standards in the context of UML are the *OMG Meta Object Facility (MOF) Core Specification, Version 2.4.2* (OMG 2014b), the *XML Metadata Interchange (XMI) Specification, Version 2.4.2* (OMG 2014c) and the *OMG Object Constraint Language (OCL), Version 2.3.1* (OMG 2012). These standards have also been formally published as ISO standards.

The MOF specification provides the *MOF meta-metamodel* which can be used for defining MOF-based metamodels such as the UML metamodel. It is also used for defining the MOF meta-metamodel itself, the elements of the MOF meta-metamodel being based on the metalanguage core defined in the UML Infrastructure specification, as is illustrated in figure 2.9. Above that, the MOF specification defines a framework for managing metadata (i. e. models) in software tools, data warehouses, repositories and similar systems.

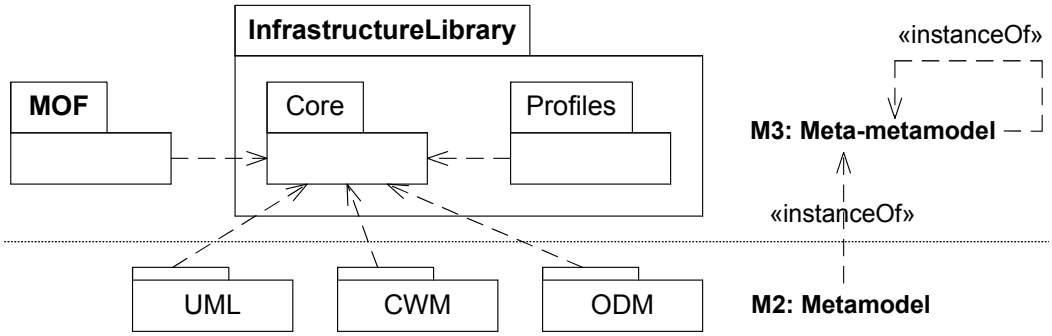


Figure 2.9: Relationship between MOF, the UML Infrastructure and MOF-based metamodels (Hitz et al. 2005, modified)

The XMI specification defines an *XML-based transfer format* which provides the basis for exchanging MOF-based metadata between heterogeneous systems. The specification defines rules for mapping MOF-based metamodels to XML Schema documents and MOF-based models to XML instance documents. In this way, MOF-based models such as UML data models can be encoded in the form of XMI documents and be exchanged between different UML tools. In practice, however, the exchange of XMI documents between different UML tools is still possible to a limited extent only, although making progress due to the work of the OMG Model Interchange Working Group which focuses on testing and identifying existing heterogeneities (Elaasar and Labiche 2012). Reasons for the limited exchangeability are that the mapping rules are complex and, thus, lead to an inconsistent implementation by different UML tools, but also that the mapping rules provide options allowing for differing implementations. Furthermore, the models can be encoded based on different versions of the XMI specification. XMI documents can also contain additional information which is not part of the model itself (cf. listing D.2 in appendix D.3). This usually comprises diagrammatic or tool-specific information such as the size or the date of creation of the individual model elements in a UML diagram which is stored in an extension section within the XMI document (Lundell et al. 2006; Elaasar and Labiche 2012).

The OCL specification provides a *formal language for defining expressions* on MOF-based metamodels and models. When defining, for instance, a UML model, often not every kind of information can be expressed within the UML model itself to make the UML model precise and unambiguous. Imagine a UML model which contains a class *Building* with the attributes *id*, *dateOfConstruction* and

dateOfDemolition. The class is to be provided with expressions that each building must have a unique identifier and that the date of construction must be before the date of demolition. These expressions could be added to the UML model in the form of a note written in natural language. However, this approach can result in ambiguities as the note could be misinterpreted by human readers; furthermore, these expressions are not machine-interpretable. By using OCL, these expressions can be added to the UML model in a precise and unambiguous way. OCL expressions are machine-interpretable, i. e. they can be validated by suitable tools and the UML model can be checked for consistency with the applied OCL expressions; and, they can also not be misinterpreted by human readers. OCL expressions can be used to define invariants, an invariant being 'a boolean expression that states a condition that must always be met by all instances of the type for which it is defined' (Warmer and Kleppe 2003). Above that, OCL expressions can be used to define the *initial value* of an attribute or association, to define *derivation rules* for how to derive the value of a derived attribute or association, to define *body expressions* which specify the result of a query operation or to define *pre-* and *postconditions* on methods (Warmer and Kleppe 2003).

3 Fundamentals of model-driven transformation of geospatial data

Besides defining geospatial data models which describe how geospatial data are to be structured, in the geospatial domain also often the necessity exists to provide geospatial data based on data models which differ from the original data structures defined for them. In order to be able to provide geospatial data based on a different data model, transformations have to be executed on the geospatial data. In particular model transformation and information integration are of importance here, both of them being rather young fields of activity which have their origin in computer science. Model transformation emerged from the discipline of Software Engineering, whereas information integration is rooted in the discipline of Databases.

Similar to chapter 2, the aim of this chapter is to establish a common terminological basis related to model-driven transformation of geospatial data in general and also to the explanations and analyses provided in the subsequent chapters of this thesis. The definitions and explanations make use of those standards relevant to the transformation of geospatial data. The chapter starts with a general introduction to the concept of *model transformation* as used in Software Engineering in general and in the OMG Model Driven Architecture approach in particular, followed by an introduction to the concept of *information integration*. Afterwards, an explanation of how these concepts are used in the geospatial domain to execute *model-driven transformation of geospatial data* is provided. The chapter concludes with a description of transformation languages relevant in the geospatial domain, in particular focusing on the *Atlas Transformation Language* (ATL), which is used for model transformation, and *UML Transformations* (UMLT), which is used for information integration, both of them being relevant in the context of this thesis.

3.1 Model-driven transformation in Software Engineering

During the last 20 years several techniques evolved which apply a model-driven approach for developing systems or software by transforming models (preferably automatically) into other models or into source code. These techniques are referred to as Model-Driven Engineering (MDE), Model-Driven Software Development (MDSD) or Model-Driven Development (MDD). By means of these techniques new models can be created from existing models and, furthermore, also source code, data (transfer) format schemas or data. All these techniques have the general concept of model transformation in common, which will be presented in the following. Also the OMG Model Driven Architecture is based on this concept, but adapts it to the use of MOF-based metamodels and models, as will be explained afterwards.

3.1.1 General concept of model transformation

Model transformation is the most integral part of all model-driven development techniques. Figure 3.1 depicts the basic concepts of model transformation between one source and one target model. The

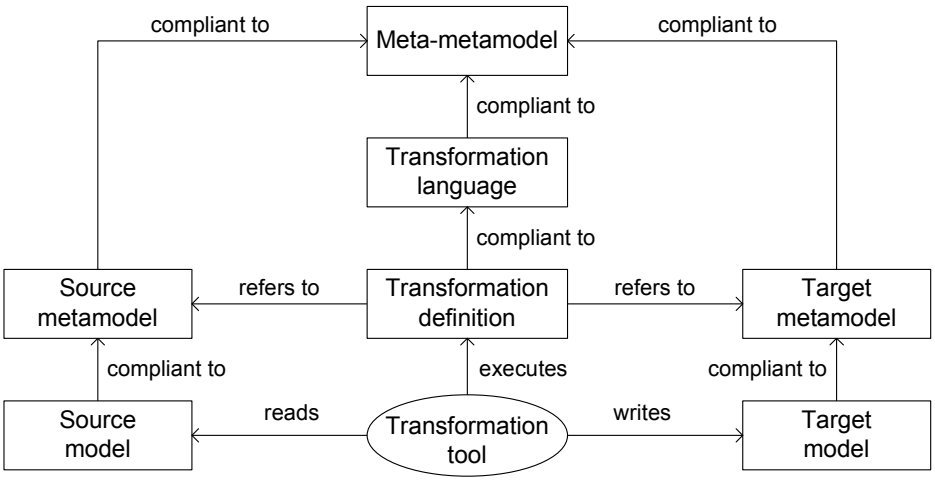


Figure 3.1: Model transformation – basic concepts (Czarnecki and Helsen 2006, modified)

transformation is defined by means of a *transformation definition* which is written using a *transformation language*. (Kleppe et al. 2003) define a transformation definition as 'a set of transformation rules that together describe how a model in the source language can be transformed into a model in the target language'; a *transformation rule*, in turn, 'is a description of how one or more constructs in the source language can be transformed into one or more constructs in the target language'. The transformation definition always refers to the metamodels of the source and the target model, thus, the *source model* has to comply with its *source metamodel* and the *target model* with its *target metamodel*. Source and target metamodel may be identical, but they may also differ from each other; both, however, have to conform to the same *meta-metamodel*. The transformation itself is executed by means of a *transformation tool*. The transformation tool reads the source model and applies the transformation definition on this model, creating in this way the desired target model. Model transformations are not restricted to exactly one source and one target model, they can also take place between one source model and multiple target models, e. g. to create several platform-specific models from one platform-independent model (cf. section 3.1.2), or between multiple source models and one target model, e. g. to combine several source models into one merged target model (Mens and Gorp 2006).

Model transformations can be classified according to various criteria (Mens and Gorp 2006; Stahl and Völter 2006). The most important criteria are:

- *Model-to-model/model-to-code transformation*: A model-to-model transformation transforms a source model into a target model, whereas a model-to-code transformation transforms a source model into artefacts such as source code, data (transfer) format schemas or documents.
- *Vertical/horizontal transformation*: A vertical transformation is executed between models located at different levels of abstraction (used e. g. in refinement and reverse engineering); a horizontal transformation, in contrast, is executed between models located at the same level of abstraction (used e. g. in refactoring, migration and model evolution).
- *Endogenous/exogenous transformation*: When the source and target model are based on the same metamodel, one speaks of an endogenous transformation (e. g. refactoring, refinement and model evolution belong to this category), whereas an exogenous transformation involves a source and a target model based on different metamodels (this holds e. g. for migration and reverse engineering).

- *Unidirectional/bidirectional transformation*: A unidirectional transformation can only be executed in one direction, i. e. from the source model to the target model; in contrast, a bidirectional transformation can be performed in two directions, i. e. from the source model to the target model and vice versa.
- *Syntactic/semantic transformation*: A syntactic transformation only changes the syntax of the model (used e. g. in data conversion), whereas a semantic transformation also takes the semantics of the model into account (used e. g. in migration and reverse engineering).

Another criterion to be considered in model transformation is the *technical space* of the source and target model (Mens and Gorp 2006). The technical space represents 'a working context with a set of associated concepts, body of knowledge, tools, required skills, and possibilities' (Bézivin 2006). The technical space is defined by the meta-metamodel (cf. section 2.3.1, page 12); thus, when the source and target model belong to different technical spaces, not only their metamodels differ, but also their meta-metamodels. Examples for technical spaces (and their corresponding meta-metamodels) are the OMG/MDA technical space (MOF), the EMF technical space (Ecore), the XML technical space (XML Schema) and the Java technical space (EBNF) (Bézivin 2006).

3.1.2 The OMG Model Driven Architecture

The *Model Driven Architecture* (MDA) is a framework for system development which follows the principles of MDE (Bézivin 2005), thus, models form the basis of the system development process. In the MDA terminology, a *system* is understood as 'any arrangement of parts and their interrelationships, working together as a whole [...] such as an entire enterprise, a process, information structures or I.T. systems' (OMG 2014a). With respect to geospatial data modelling, geospatial data represent based on this definition such information structures and, thus, the system, which is described by a geospatial data model. MDA was developed by the OMG and comprises a set of standards relevant for the implementation of MDA. Among these standards are MOF (cf. section 2.4.4, page 25), which provides the core concepts for defining MOF-based metamodels, UML (cf. section 2.4, page 18), which can be considered the primary MOF-based metamodel for defining models to be employed in the MDA process, and XMI (cf. section 2.4.4, page 25) for exchanging UML models in the form of XML-based documents. Further standards are the transformation languages MOF Query/View/Transformation (QVT) (OMG 2011a) for defining transformations between models (cf. section 3.4.3) and MOF Model to Text Transformation Language (M2T) (OMG 2008) for transforming models into textual representations such as source code or documents.

The MDA approach supports the implementation of separation of concerns, in particular 'the separation of the business concerns of a system from the technology-dependent implementation concerns of components of that system' (OMG 2014a). When specifying, for instance, a software system using high-level diagrams, the business processes of that system are often mixed with technical information. MDA, however, clearly separates these two aspects by making use of three different models, as is displayed in figure 3.2, which represent the system from the following viewpoints (Kleppe et al. 2003):

- *Platform Independent Model* (PIM): This model describes the system to be developed at a high level of abstraction, i. e. it defines the business logic of the system independent of any particular technology platform the implemented system might be deployed later on and is, thus, referred to as being *platform-independent*. In general, the MDA process is always initiated by defining a PIM of the system to be developed.

- *Platform Specific Model* (PSM): A PSM is a model which describes the system at a lower level of abstraction than the PIM, as it takes into account the technical aspects of the specific technology platform on which the system will run later on and, thus, is *platform-dependent*. A PSM is generated by transforming a PIM into a PSM based on a predefined transformation definition. Since one system can be deployed on several different technology platforms, it is also possible to transform one PIM into several different PSMs, each one targeted at a specific platform. To obtain interoperability between the individual PSMs, the MDA approach provides for so-called bridges which can be generated in addition to the PSMs. Often, PIMs as well as PSMs are defined using UML; in this case, the PIM UML model only defines the platform-independent concepts, whereas the PSM UML model also contains the platform-specific concepts. One possibility of introducing platform-specific concepts to a UML model is to define and apply a corresponding UML profile (cf. section 2.4.1, page 19)[1].
- *Platform Model* (PM): This model represents the source code of the implemented system and is generated by transforming a PSM based on a predefined transformation definition. Since the source code can be considered an abstraction of the machine code which is generated from the source code by a compiler, also the source code is regarded as a model in MDA.

When defining a transformation specification, it has to be taken care, that the semantics of the source model are preserved in the target model. The transformation of a PIM into a PSM is considered to be more complex than the transformation of a PSM into a PM, as the former step can, depending on the platform the PSM is to be targeted at, impose an extensive modification on the structure of the model, whereas the latter step simply maps a model conforming to a certain platform into the corresponding platform code (Kleppe et al. 2003).

The MDA-based system development process comprises a sequence of transformation steps and – what characterises the MDA approach in particular – in each step formal, i. e. machine-interpretable, models are created. Another specific characteristic of MDA is that the transformation from a PIM to a PSM can be fully automated by using transformation tools since the PIM represents a machine-interpretable model. Traditionally, models for specific platforms had to be created manually from high-level diagrams. Both, machine-interpretable models and automatic transformation, require that the models are created using a formal language (cf. section 2.2.1, page 9).

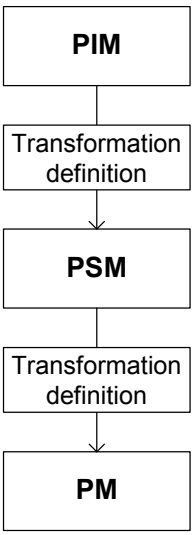


Figure 3.2: The MDA concept

As regards the classification criteria mentioned in the previous section, they can be applied to MDA as well. Examples of model-to-model transformations are PIM → PIM (horizontal transformation) and PIM → PSM (vertical transformation), whereas PSM → PM can be considered as a model-to-code transformation (vertical transformation). PIM → PIM transformations can, furthermore, be endogenous or exogenous, depending on whether the source and target PIMs are based on the same metamodel or not, whereas PIM → PSM transformations are usually endogenous. The technical space, however, is identical for all source and target models since the MDA approach requires the metamodel of each PIM model and of each PSM model to be based on the meta-metamodel MOF.

[1] It should be noted, however, that a UML model to which a UML profile is applied does not automatically always represent a PSM. A UML model with applied UML profile can just as well represent a PIM. It rather depends on the kind of concepts provided by a specific UML profile, whether they are platform-independent or platform-specific.

This section is to be concluded with a small example of a possible MDA transformation workflow within the geospatial domain. As described above, one PIM can be transformed into several PSMs, each PSM targeted at a different platform. In the context of a specific geospatial application, cadastre information shall be made usable by means of different platforms; amongst others, the information shall be transferable using the data transfer formats GML and GeoJSON, be processable via the programming language Java and be storable in a relational database. The required workflow is shown in figure 3.3: First, a suitable PIM containing the relevant cadastre information is defined, which afterwards is transformed into PSMs for the platforms GML, GeoJSON, Java and Relational. Finally, the generated PSMs are transformed into the corresponding formats and code. To be able to execute the transformations, individual transformation definitions are required for each platform and for each transformation step.

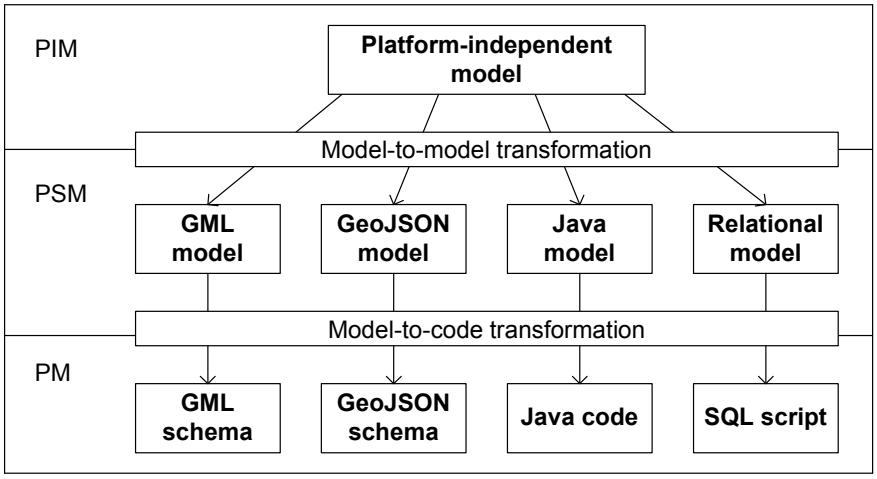
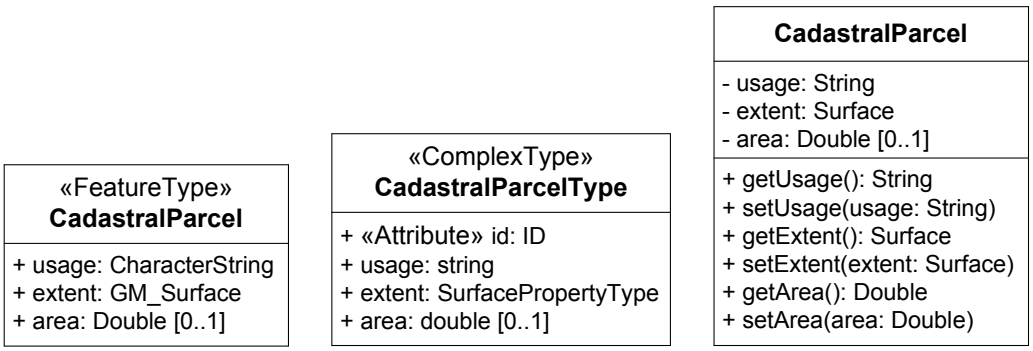


Figure 3.3: Example MDA transformation workflow

Figure 3.4 zooms in on the imaginary model for cadastre information and illustrates how the workflow affects a fictitious feature type *CadastralParcel*. In the PIM, the feature type is represented by a UML class to which the stereotype «FeatureType» (cf. section 5.2, page 85) is applied, as is shown in figure 3.4(a). Figures 3.4(b) and 3.4(c), in turn, display the corresponding UML classes for the platforms GML and Java, respectively, which result from the transformation of the PIM into GML-specific and Java-specific PSMs.

In a GML application schema, a feature type is usually represented by an XML element declaration whose name is identical to the name of the UML class and by an XML complex type definition with the same name as well, but which is extended by the suffix *Type*[2]. In addition, the XML element is provided in GML with an XML attribute named *id*. These XML/GML-specific concepts can be added to the PSM representation of the UML class *CadastralParcel* by means of a UML profile which defines these concepts. The *UML Profile for XML Schema* (Carlson 2008) is such a UML profile which provides stereotypes to be used for generating XML Schema documents from UML models. Based on this UML profile, the UML representation of the UML class *CadastralParcel* can be targeted to the GML platform by explicitly denoting the UML class as XML complex type through

[2]Further UML-to-GML encoding rules for feature types defined in the standard ISO 19136 Annex E (cf. section 5.3, page 86) are not taken into account in this example.



(a) PIM (b) GML PSM (c) Java PSM

Figure 3.4: PIM and PSM representations of an example UML class (Kleppe et al. 2003, modified and extended)

the stereotype «ComplexType» and by adding the UML attribute *id* to the UML class, characterising it as XML attribute through the stereotype «Attribute».

The UML attributes of a PIM UML class are usually declared public. 'The meaning of a public attribute in a PIM is that the object has the specified property, and that this property can change value over time' (Kleppe et al. 2003). Java, however, supports the object-oriented concept of encapsulation, which means that within a Java class the Java attributes should be declared private such that they are only accessible via equivalent Java get and set methods. Thus, the transformation definition for the Java-specific PSM needs to create for each public UML attribute in the PIM model an identical private UML attribute as well as public UML get and set operations in the corresponding UML class of the Java PSM model (Kleppe et al. 2003). Afterwards, the two PSM UML classes depicted in figures 3.4(b) and 3.4(c) can easily be mapped to the corresponding GML application schema or Java code, respectively.

3.2 Information integration through transformation

Model transformation defines transformations *between source and target metamodels* and determines in which way metamodel concepts used in the source model(s) are to be transformed into metamodel concepts used in the target model. *Information integration*, in contrast, defines transformations *between source and target models* and determines in which way the contents of the source data compliant to the structure and semantics of the source model(s) are to be transformed into target data compliant to structure and semantics of the target model.

Data models describe the structure of data; however, since the process of modelling is always influenced by the perception of a particular modeller (cf. mapping characteristic in section 2.1.1, page 5), different modellers will usually come up with different models, defining the structure of the data in differing ways. Furthermore, whether certain information is captured by the data model or not, usually depends on the application a data model is intended for. Thus, data models covering the same universe of discourse can be structured in different ways and can contain different information. These and other differences can result in data models which exhibit various types of heterogeneity. These heterogeneities, in turn, can lead to problems in information integration and need to be addressed by

different types of transformation. Both, the different types of heterogeneity and the different types of transformation resulting therefrom, will be presented in the following.

3.2.1 Types of heterogeneity

In general, in the context of information integration different types of heterogeneity can be distinguished. The description provided here follows the classification given by (Leser and Naumann 2007) which differentiates the following five types of heterogeneity:

- *Technical heterogeneity*: This type of heterogeneity does not refer to data models and the structure of data, but to the way of accessing data provided by a system. This includes the method of requesting data from a system (e. g. by means of a query language when requesting data from a database or by means of parametrised functions when requesting data from a web service), the query language used for retrieving the information of interest (e. g. SQL), the transfer format providing the requested information (e. g. XML) and the communication protocol able to communicate with the system (e. g. HTTP). In the geospatial domain system heterogeneity can be resolved by applying relevant ISO and OGC standards which allow for requesting data in the form of standardised data transfer formats through standardised geospatial web service interfaces.
- *Syntactic heterogeneity*: This type of heterogeneity addresses technical differences regarding the representation of identical information such as the character encoding (Unicode or ASCII) or the kind of delimiter used in CSV files. Syntactic heterogeneity is not considered as problematic in information integration.
- *Data model heterogeneity*: Data model heterogeneity occurs when identical information is provided in the form of different data models. The term data model corresponds here to the term modelling paradigm as introduced in section 2.2.2, page 10. The information of the INSPIRE data specifications, for instance, is provided in the form of an object-oriented data model (OO paradigm) when defining the structure of the data conceptually, whereas in database management systems the information exists in the form of a relational data model (Relational paradigm). Furthermore, when using GML as data transfer format, the information takes the form of a joint XML/OO data model, since GML combines the XML paradigm with the OO paradigm. Data model heterogeneity often comes along with semantic heterogeneity (see below), since the concepts provided by the different modelling paradigms own specific semantics which are determined by each modelling paradigm separately. In the OO paradigm, for instance, classes can be related to each other through inheritance, in the Relational paradigm, however, this concept is not supported. As regards the general concept of model transformation (cf. section 3.1.1, page 27), data model heterogeneity can occur in vertical transformation, when the models located at different levels of abstraction are based on different modelling paradigms, and in horizontal transformations, when the technical spaces of the source and target models differ, i. e. when their meta-metamodels comply to different modelling paradigms.
- *Structural* and *schematic heterogeneity*: Structural heterogeneity occurs when two models differ from each other, although they describe the same universe of discourse and are based on the same modelling paradigm. A simple example for structural heterogeneity is depicted in figure 3.5. The UML classes *Facility* and *Building* both represent buildings and allow for storing the address of a building. The UML class *Facility* provides specific attributes for storing the street name, the house number, the postal code and the city of the building, whereas the UML class *Building* provides only one attribute which combines all these pieces of information. Thus, the structure regarding the address information differs between the two UML classes, the semantics, however, is identical.

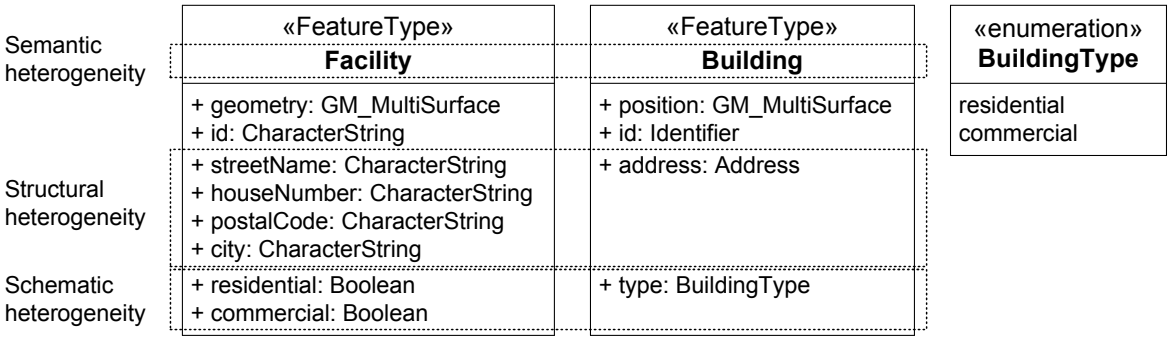


Figure 3.5: Example of structural, schematic and semantic heterogeneity

Schematic heterogeneity is often considered a special case of structural heterogeneity. Schematic heterogeneity occurs when different metamodel concepts are used for modelling the same kind of information. However, it is not always possible to separate structural from semantic heterogeneity, often both types of heterogeneity coincide. As regards the example in figure 3.5, the UML class *Facility* defines the two attributes *residential* and *commercial* for being able to classify a building as residential or as commercial building (UML metaclass *Property*), whereas the UML class *Building* makes use of corresponding enumeration literals defined in the enumeration *BuildingType* (UML metaclass *Enumeration*). A third possibility would be to define two separate UML classes, the first one named *ResidentialBuilding* and the second one named *CommercialBuilding*, in order to express the desired information (UML metaclass *Class*). The use of these different metamodel concepts results in schematic differences in the depicted UML classes, but at the same time also in structural differences.

- *Semantic heterogeneity*: Semantics deals with the meaning, i. e. interpretation, of the modelled information (cf. section 2.2.1, page 9). This type of heterogeneity occurs, amongst others, when the information defined in two models has been given a different name, but has the same meaning (synonym), or when the information is named identically, but its meaning differs (homonym). An example for semantic heterogeneity in the form of a synonym is given in figure 3.5. The UML class on the left side is named *Facility*, whereas the UML class in the middle is named *Building*. Both UML classes, however, have the same meaning and model the same kind of information. An example for a homonym is the word *bank* which denotes, on the one hand, a financial institution and, on the other hand, the shore of a river. A correct interpretation of the modelled information requires knowledge about the area of application the model was defined for. Geospatial data models are, thus, often accompanied by a *feature catalogue* which provides a more detailed description of the modelled information in order to prevent its misinterpretation by the data model user.

In geographic information science literature, heterogeneities are defined similarly, distinguishing between system (Sheth 1999; ISO 2014), syntactic, structural/schematic and semantic heterogeneity (Bishr 1998; Sheth 1999; ISO 2014). However, the terminology can vary; for instance, the definition for syntactic heterogeneity given in (Bishr 1998) rather refers to data model heterogeneity according to the above description. Furthermore, an aspect specific to geographic information is mentioned there, namely that heterogeneity can also occur with respect to the geometric representation of spatial objects as raster or vector data, which also belongs to data model heterogeneity. In fact, the geometric representation rather would require an examination of its own since this topic contains many more aspects than simply a division in raster and vector data. Considering vector representation alone,

objects can be modelled in different levels of detail; a building, for instance, could be modelled geometrically as point, as polygon or as solid. For representing solids, in turn, various modelling paradigms exist, such as Boundary Representation (B-rep), Constructive Solid Geometry (CSG) or scene graphs (Nagel et al. 2009).

3.2.2 Types of transformation in the context of information integration

The different types of heterogeneity described above require different types of transformation to achieve information integration. Figure 3.6 depicts the relationship between the different types of heterogeneity and the different types of transformation resulting therefrom. The following types of transformation exist:

- *Syntactic transformation*: This type of transformation changes the syntax of the data; this means that a conversion between different data (transfer) formats is conducted which can also involve a conversion between different data models (modelling paradigms). An example is the transformation of data provided in a database or as Shapefile (ESRI 1998) document (Relational paradigm) into a GML document (XML/OO paradigm), which e. g. is the case when requesting data from an OGC Web Feature Service (WFS). By means of syntactic transformation, technical, syntactic and data model heterogeneity can be solved.
- *Semantic transformation*: This type of transformation modifies the structure of the data in such a way that it complies to the semantics of a different data model. Semantic transformation solves structural/schematic and semantic heterogeneity issues. An example is the transformation between the classes *Facility* and *Building* from figure 3.5. According to the level at which the transformation is defined, semantic transformation can be further classified into format-driven, format-schema-driven and model-driven transformation (cf. section 3.3.2, page 39, for their detailed description).

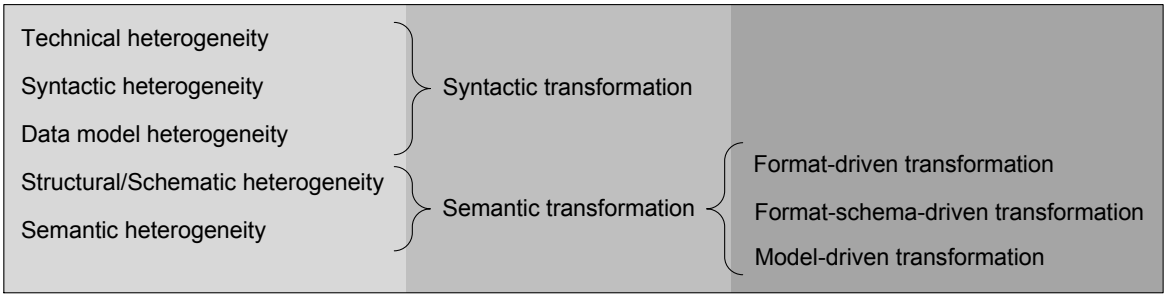


Figure 3.6: Relationship between the different types of heterogeneity and transformation

In geographic information science literature, also a classification into three types of transformation can be found which extends the above classification by a so-called *schematic transformation* such as in (Lehto 2007b). In this case, the semantic transformation solely deals with the semantic heterogeneity aspects, whereas the schematic transformation focuses on the schematic heterogeneity aspects.

By means of information integration source data compliant to the structure and semantics of the source model(s) are transformed into target data compliant to the structure and semantics of the target model. Two types of information integration exist, *virtual integration* and *materialised integration* (Leser and Naumann 2007). Using virtual integration, the transformed data are not stored permanently in the target system; each time, a user requests transformed data, the transformation has to be executed anew. Materialised integration, in contrast, means that the transformed data are stored permanently in the target system (cf. section 3.3.2, page 39, for their use in the geospatial domain).

Information integration can be divided into two phases, the configuration phase and the execution phase. The configuration phase is also referred to as *schema mapping*. Schema mapping deals with relating elements of the source and target models by means of correspondences; in this phase the transformation definition is created. Afterwards, the *execution phase* follows which carries out the transformation based on the previously defined transformation definition (Lehto 2007b).

In the context of information integration the source models are usually called *local schemas*, whereas the target model is referred to as *global schema* (Lehto 2007a). The global schema shall contain all concepts of the different local schemas (Leser and Naumann 2007). Another term commonly used is *view*. In contrast to the global schema, a view usually contains only those pieces of information from the data model which are of interest to a user in the context of a certain area of application (Kemper and Eickler 2013).

Data warehouses represent a system with materialised integration. The process of integrating data from one or several source systems into a data warehouse is referred to *ETL*, which is an acronym for *Extract, Transform, Load*. The process involves the following steps: (1) Extraction of the data from the relevant source systems, (2) transformation of the extracted data such that they are compliant to the schema of the data warehouse and (3) loading and storage of the transformed data in the data warehouse (Leser and Naumann 2007). In the geospatial domain also the term *Spatial ETL* exists; Spatial ETL denotes ETL processes which involve the integration of geospatial data (Safe Software 2015c).

3.3 Model-driven transformation in the geospatial domain

Model-driven transformation of geospatial data as used in the geospatial domain is based on the concepts of model transformation and information integration presented above in this chapter. The model-driven transformation of geospatial data occurs in two different forms, as vertical transformation to execute encodings and as horizontal transformation to perform information integration. Both forms are presented in the following.

3.3.1 Encoding of geospatial data as defined in the standard ISO 19118

The term *encoding* as defined in the standard *ISO 19118 Geographic information — Encoding* (ISO 2011) denotes the process of transforming system-dependent data structures conforming to a specific application schema into system-independent data structures suitable for transfer and storage. Encoding is required for enabling the transfer of data between heterogeneous systems. ISO 19118 defines a concept for how to transfer data between two systems which is displayed in figure B.1, page 175. The concept assumes that the semantic structures of the data from the source and target systems are defined by internal schemas and that also the syntactic structures of the data are defined internally by the source and target systems. To enable data transfer, the data from the source system have to be transformed in such a way that, on the one hand, their semantics conforms to the semantics of a specific external schema, called *application schema* (cf. section 2.3.2, page 14), and that, on the other hand, their syntax conforms to a certain system-independent *data transfer format* suitable for exchanging data between different systems and whose structure is defined by a *data transfer format schema*.

The transformation is defined by a so-called *encoding rule* which is ‘an identifiable collection of conversion rules that defines the encoding for a particular data structure’ (ISO 2011). Each encoding

rule consists of *schema conversion rules* which define a transformation between the application schema and the data transfer format schema and of *instance conversion rules* which define a transformation between instances from the source system and corresponding instances in the data transfer format. Focusing on the individual transformations, the complete workflow of data transfer is composed of the following steps as is also depicted in figure 3.7:

1. Transformation T_{AT} of the application schema into a data transfer format schema which defines the structure of the data transfer format for the data to be transferred.
2. Transformation T_{IA} of the system-dependent source data $d_{sourceI}$ whose semantics is compliant to the internal schema of the source system into system-dependent source data $d_{sourceA}$ such that their semantics conforms to the semantics of the application schema. This is a semantic transformation; the syntax of the data still conforms to the syntax of the source system.
3. Transformation T_{SDSI} of the system-dependent source data $d_{sourceA}$ into system-independent transfer data $d_{transferA}$ such that not only the semantics of $d_{sourceA}$ conforms to the semantics of the application schema, but that also their syntax conforms to the syntax of the chosen system-independent data transfer format. This is a syntactic transformation; the semantics is not altered.
4. Transfer of the data from the source system to the target system.
5. Transformation T_{SISD} of the system-independent transfer data $d_{transferA}$ into system-dependent target data $d_{targetA}$ such that their syntax conforms to the syntax of the target system. This is a syntactic transformation; the semantics are not modified, which means that $d_{targetA}$ remains compliant to the semantics of the application schema.
6. Transformation T_{AI} of the system-dependent target data $d_{targetA}$ whose semantics is compliant to the application schema into system-dependent target data $d_{targetI}$ such that their semantics conforms to the semantics of the internal schema of the target system. This is a semantic transformation; the syntax remains unchanged.

The schema conversion rules are executed by the transformation T_{AT}, the instance conversion rules by the transformations T_{SDSI} and T_{SISD}. The transformations T_{IA} and T_{AI} are not part of the encoding rule, they represent internal processes within the source and target systems, respectively.

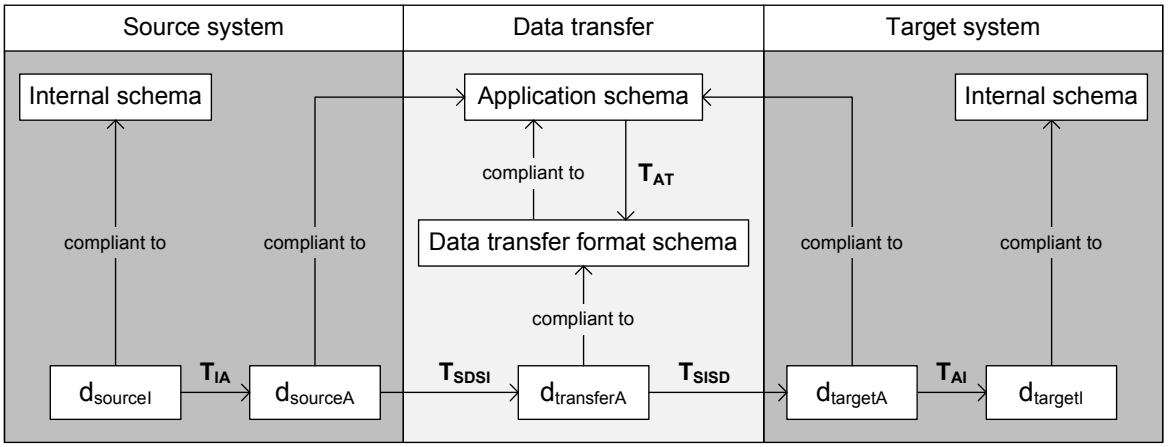


Figure 3.7: Required transformations during ISO-19118-based data interchange between two systems

The application schema and the system-independent data transfer format enable both systems to interpret the semantics and the syntax of the data in the same way. According to the standard ISO 19118, applications schemas are to be defined using UML in compliance with the standards ISO/TS

19103 and ISO 19109. A widely used data transfer format for geographic information is the Geography Markup Language (GML), an XML-based language for modelling, transferring and storing geospatial information, which is defined by the standard ISO 19136 together with an encoding rule for mapping UML application schemas to GML application schemas (cf. section 5.3, page 86). Other commonly used transfer formats are, for instance, Shapefile, GeoJSON, KML, GeoTIFF or CityGML (Portele 2012a).

As regards the MDA approach described in section 3.1.2, page 29, the transformation T_{AT}, which comprises the schema conversion rules, corresponds to a direct PIM $\rightarrow$ PM transformation; it does not generate a PSM as intermediate step in the encoding process, as is shown in the left part of figure 3.8. The encoding rule for this transformation is defined between the concepts of the application schema language and the concepts of the data transfer format schema language. The encoding rule does not refer to a specific application schema, allowing in this way for defining the encoding rules application-schema-independent such that they 'can be used for different application schemas as long as the schemas are defined in the same conceptual schema language' (ISO 2011) and as long as the same data transfer format schema is used. However, to be able to transform directly from a PIM into a PM, the PIM either already needs to include all platform-specific aspects, which means the PIM is not entirely platform-independent any more, or the encoding rule needs to cover all aspects – from the transformation definition between PIM and PSM to the conversion rules between PSM and PM – to be able to bypass the PSM.

To comply with the PIM $\rightarrow$ PSM $\rightarrow$ PM transformation approach defined by the MDA framework, the transformation T_{AT} can be extended by making use of an *implementation schema* (cf. section 2.3.3, page 16) as intermediate step in the encoding process, which is depicted in the right part of figure 3.8. The implementation schema represents the PSM and, thus, includes all platform-specific aspects of the data transfer format schema to be generated. In this way, first a PIM $\rightarrow$ PSM transformation between the application schema and the implementation schema is executed, followed by a PSM $\rightarrow$ PM transformation to generate the data transfer format schema from the application schema. This approach is applied in Germany for deriving the German data transfer format schema NAS from the German AAA reference model (AdV 2009).

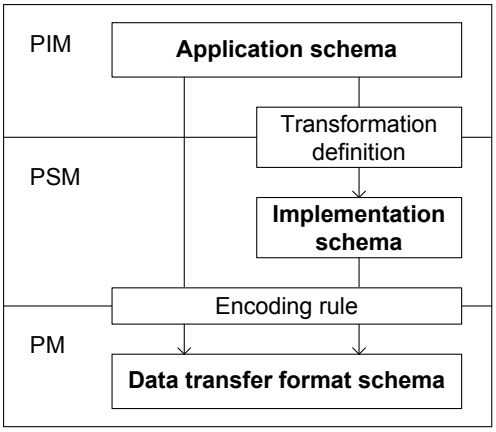


Figure 3.8: ISO-19118-based encoding of application schemas in the context of the MDA approach

When creating schema encoding rules using either of the above approaches, attention has always to be paid to the semantics defined by the application schema of the data to be transferred. During the encoding process this semantics is not allowed to be changed neither in the implementation schema nor in the data transfer format schema, the schemas are only allowed to be complemented by platform-specific details.

The transformation T_{SDSI}, which comprises the instance conversion rules, corresponds in terms of MDA to a PM $\rightarrow$ PM transformation. Since the source data $d_{sourceA}$ are already compliant to the semantic concepts of the transfer data $d_{transferA}$ (cf. figure 3.7), the transformation solely needs to convert the data instances from a system-dependent syntactical representation into a system-independent syntactical representation and, thus, represents a syntactic transformation. The same

holds for the transformation T_{SISD}. ISO 19118 states that for a successful data transfer the source and target system have to agree on an application schema, an encoding rule and a transfer protocol. However, as regards the encoding rule to decide upon, the agreement only needs to cover the schema conversion rules for generating the required data transfer format schema. The instance conversion rules are specific to the source and target systems the data are stored in. The target system does not need to be aware of the instance conversion rules required by the source system, as they are of no value to the target system, and vice versa.

3.3.2 Information integration of geospatial data

The process of data interchange presented above transforms data compliant to the internal schema of the source system into data compliant to the internal schema of the target system. The process involves the transformation steps T_{IA} and T_{AI} which represent semantic transformations as they transform the data between schemas with heterogeneous semantics, i. e. between the internal schemas of the source and target systems and the application schema, and, thus, have to take into account the semantics of the internal schemas and of the application schema. However, as regards the definition and execution of these transformations, ISO 19118 only states that 'this is done by defining a mapping from the concepts of the internal schema to the concepts defined in the application schema and by writing appropriate mapping software to translate the data instances' (ISO 2011). The standard does not give advice at which level the mapping should be defined; also, the above concept does not consider data format schemas for the source data $d_{sourceI}$ and the target data $d_{targetI}$.

In general, mappings between source and target data can be defined at three different levels, the data format level, the data format schema level and the conceptual schema level, resulting in a semantic transformation which can be classified as format-driven transformation, format-schema-driven transformation and model-driven transformation (Fichtinger 2011; Kutzner, Schilcher et al. 2012), respectively, as is also shown in figure 3.6:

- At the *data formal level* the mappings are defined between two specific data formats, i. e. a source data format and a target data format. The transformation tool reads these mappings and transforms the source data format directly into the target data format. Thus, the transformation which is executed based on these mappings can also be referred to as *format-driven transformation*. This is exemplified by figure 3.9. The data format DF_x which is compliant to the source model M_{source} is to be transformed into the data format DF_y which is compliant to the target model M_{target}. DF_x and DF_y can be derived from M_{source} and M_{target} by making use of the encoding rules ER_x and ER_y, respectively, as introduced in the previous section. The mappings between the concepts of M_{source} and M_{target} are defined within a transformation definition which directly refers to DF_x and DF_y. A transformation tool reads this transformation definition and transforms DF_x directly into DF_y. One disadvantage of this approach is that individual transformation definitions need to be created for each differing combination of source and target data formats since the mappings are always defined between two specific formats. Another disadvantage is that a modification of the source and/or target model involves a re-derivation of all corresponding data formats to remain compliant to their model and, consequently, a readjustment of each transformation definition individually to match the modified data formats. Furthermore, in addition to knowledge about the modelled objects itself, knowledge about the data formats is required when defining the mappings at the data format level.
- At the *data format schema level* the mappings are defined between specific data format schemas; the transformation tool uses these mappings to transform the data formats which are based on these data

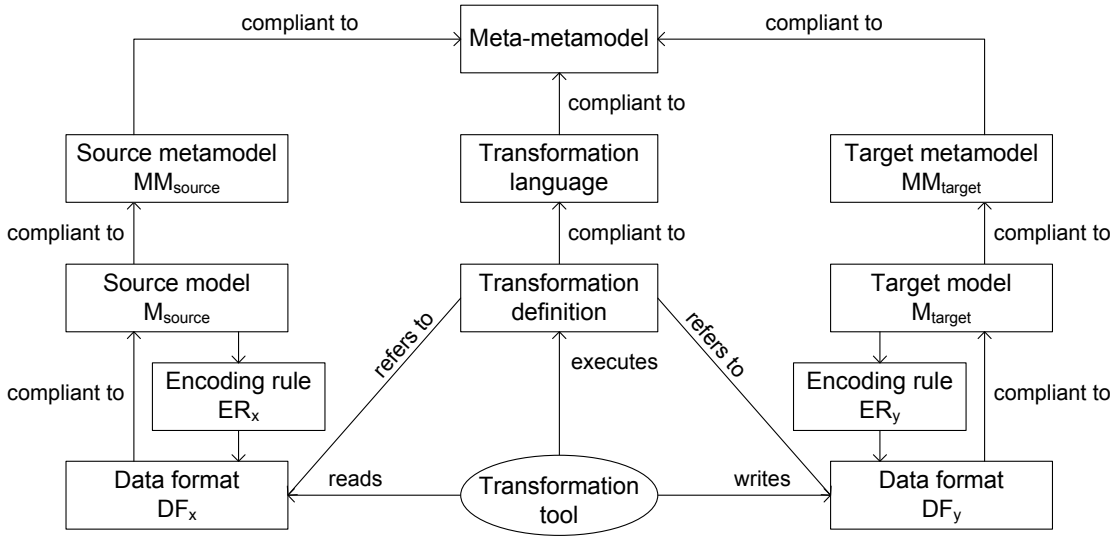


Figure 3.9: Format-driven transformation of geospatial data

format schemas. This transformation is also referred to as *format-schema-driven transformation*. Apart from defining the mappings one level above the data format level, this approach features the same characteristics as the format-driven transformation.

- At the *conceptual schema level* the mappings are defined between one or several source models and one target model; the transformation tool reads these mappings and executes a transformation between the corresponding source data format and target data format. Thus, the transformation which is executed based on these mappings is referred to as *model-driven transformation*. Similar to the general concept of model transformation as described in section 3.1.1, page 27, the mappings are defined here as well at the conceptual schema level. However, one decisive distinction exists: The transformation definition used in the general concept of model transformation is defined between source and target metamodels and determines in which way metamodel concepts used in the source model are to be transformed into metamodel concepts used in the target model. The transformation definition used in model-driven transformation of geospatial data, in contrast, is defined between source and target models and determines in which way the contents of the source data format compliant to the structure and semantics of the source model(s) are to be transformed into a target data format compliant to the structure and semantics of the target model. This kind of transformation is equivalent to the concept of *information integration* which originates from the database domain. (Lehto 2007a) also speaks of *content transformation*. Figure 3.10 displays the concept. In contrast to the format-driven transformation explained above, the mappings are now defined within a transformation definition which refers to M_{source} and M_{target}. Furthermore, another important aspect is involved in the transformation, i. e. the encoding rules ER_x and ER_y for deriving DF_x and DF_y from M_{source} and M_{target} of the geospatial data to be transformed. To be able to transform the geospatial data in a correct way, it is not sufficient for the transformation tool to know in which way the contents need to be transformed; the transformation tool also needs to know according to which structure the contents are encoded in the data formats. Thus, a model-driven transformation of geospatial data can only be transformed in a correct way, when the corresponding encoding rules are taken into account by the transformation tool during the transformation. One advantage of this approach is that the transformation definition is defined

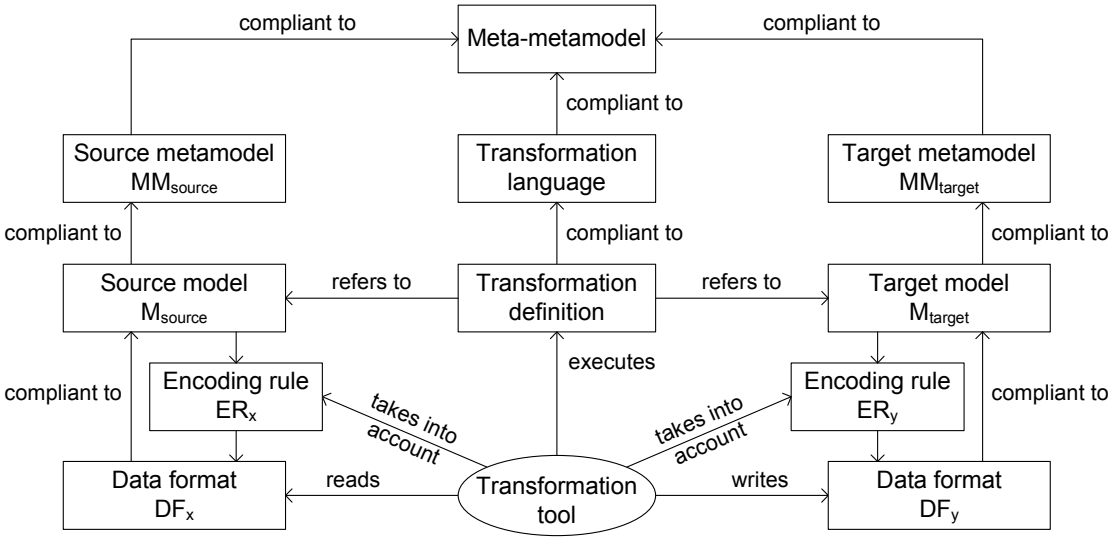


Figure 3.10: Model-driven transformation of geospatial data (Kutzner and Eisenhut 2010, modified and extended)

between the source and target models and, thus, only needs to be defined once; yet, an automatic transformation between various data formats is possible due to the specific encoding rules which are taken into account by the transformation tool. Furthermore, when the source and/or target model is modified, only this one transformation definition needs to be adjusted to match the modified model, in contrast to the format-driven transformation where several transformation definitions need to be adjusted.

All these three transformation approaches can be applied to transform geospatial data between data formats compliant to different models. Moreover, the approaches can also be applied to the transformation steps T_{IA} and T_{AI}, as described in section 3.3.1, page 36, to transform geospatial data between the internal schemas of the source and target systems and the application schema.

Model-driven transformation of geospatial data can be implemented in two different ways, on-the-fly and off-line. On-the-fly transformation is equivalent to virtual integration, whereas off-line transformation corresponds to materialised integration (cf. section 3.2, page 32). The decision regarding which of the two approaches should be used depends on various criteria, such as on the application itself, on the complexity of the models and on the user requirements (Eisenhut, Illert et al. 2010):

- *On-the-fly transformation*: This transformation is executed whenever the user requests transformed data; thus, on-the-fly transformation is suited in particular when the actuality of the transformed data is of crucial importance. Another advantage of this approach is that the data can be transformed into any target model directly at request time. However, the performance of the transformation might be affected when huge amounts of data have to be transformed or when the transformation by itself is very complex due to substantially differing source and target models (Eisenhut, Illert et al. 2010). Since the user requests data which conform to the target model, the user always defines the request according to the semantics of the target model. The transformation tool, however, requires the request conforming to the semantics of the source model, which is why the request first has to be translated by the transformation tool into the semantics of the source model, before the transformation can be carried out. On-the-fly transformation can be executed by the data

provider, but it can also by provided by means of external, third-party transformation tools such as transformation network services.

- *Off-line transformation*: This transformation is executed in advance, i. e. independent from whether the user requests data. The transformed data are stored in a database (also referred to as staging database) and the user sends the request directly to the database and also receives the transformed data from there. No transformation needs to be executed at the time of the user request and, thus, the request does not need to be translated into the semantics of the source model by the transformation tool. Off-line transformation is always an internal process on the part of the data provider. Off-line transformation is considered to be more suitable for complex models involving complex transformations in order to satisfy obligatory performance requirements since at the time of the request the transformation has already been executed and, thus, no performance loss will occur. However, since the transformation is only executed at certain intervals and then stored in a database, the actuality of the transformed data cannot be guaranteed.

3.4 Transformation languages – the medium for defining transformations

As explained above, model transformation involves transformations between source and target metamodels, whereas information integration results in transformations between source and target models. This means that two different types of transformation definitions have to be created using suitable transformation languages; on the one hand, a transformation language able to cope with metamodels is required and, on the other hand, a transformation language able to cope with models. In the following, two transformation languages will be presented in detail, *ATL* as a representative of transformation languages suitable for model transformation and *UMLT* as an example of transformation languages suitable for information integration. Both languages will be employed later in this thesis. In addition, four more transformation languages used in various projects in the geospatial domain (cf. section 4.3.1, page 63) will be introduced shortly, *QVT* as another representative belonging to the first category, *RIF* and *EDOAL/gOML* as further examples belonging to the second category and *XSLT* as a language which is applicable to both categories.

In the same way as modelling languages serve in defining models, transformation languages serve in defining transformation definitions between models. Transformation languages are formal languages and, thus, exhibit the same characteristics as modelling languages (cf. section 2.2.1, page 9). In addition, transformation languages can be classified according to various other characteristics, too, such as the type of model transformation a transformation language can be applied to (e. g. horizontal or vertical, uni- or bidirectional, cf. section 3.1.1, page 27). Another important classification criterion is based on two paradigms used for categorising programming languages, the declarative paradigm and the imperative paradigm. Transformation languages following the declarative paradigm 'focus on what needs to be transformed into what by defining a relation between the source and target models' (Mens and Gorp 2006), whereas transformation languages based on the imperative paradigm 'focus on how the transformation itself needs to be performed by specifying the steps that are required to derive the target models from the source models' (Mens and Gorp 2006). Transformation languages can also realise a hybrid approach which takes into account both paradigms.

The transformation languages presented here will be put in the context of these various classification criteria to provide some orientation; however, no evaluation based on these criteria will be conducted, as this would involve a by far more detailed analysis of transformation languages in general which is beyond the scope of this work.

3.4.1 ATL – A transformation language for model transformation

A *Request for Proposal: MOF 2.0 Query / Views / Transformations RFP* (OMG 2002) was issued by the OMG in 2002 to receive proposals for a transformation language which is based on MOF and which follows the MDA approach. ATL (Atlas Transformation Language) was one of the proposals submitted. Although it was not adopted by the OMG as transformation language specification, ATL is widely-used today. An open-source implementation of ATL is available as plug-in for the integrated development environment (IDE) Eclipse (The Eclipse Foundation 2015a). The transformation approach used by ATL is compliant to the concepts presented in sections 3.1.1, page 27, and 3.1.2, page 29, on model transformation and MDA. This means that the source and target models involved in the transformation have to be compliant to MOF-based metamodels and that also the ATL metamodel itself is based on MOF. ATL, thus, corresponds to the OO paradigm. The following description of ATL is based on (Jouault and Kurtev 2005; The Eclipse Foundation 2015b).

ATL is a hybrid transformation language, i. e. it provides declarative and imperative constructs for creating transformation definitions. However, transformations should primarily be specified using declarative constructs, imperative constructs should only be used for definitions which are difficult to express in a declarative way. ATL transformations are defined in a textual concrete syntax. The data types and declarative expressions of the ATL language are based on the OCL specification. ATL makes use of the OCL primitive types and collection types and implements a large number of OCL data type operations, but also defines additional data types and operations.

ATL provides three different units to compose an ATL transformation: transformation modules, queries and libraries. The main component of an ATL transformation is the *transformation module*; it defines the actual transformation definition between source and target models. Each transformation module specifies within a *header* the source and target models which participate in the transformation. Listing 3.1 provides an example of such a transformation module named *AAA2Core*. The listing is an excerpt from an ATL transformation defined for transforming the German AAA reference model to which the AAA UML profile (cf. section 5.5.1) is applied into a model with equivalent definitions, but to which the Core UML profile from section 6.3, page 123, is applied.

Listing 3.1: Example ATL transformation module

```
-- Header
module AAA2Core;
create OUT: UML2 from IN: UML2, COREPROFILE: UML2, STANDARDPROFILE: UML2;

-- Library import
uses "lib::UML2";
uses UML2Copy;

-- Helper
helper context UML2!Element def: hasStereotype(name: String): Boolean =
  not self.getAppliedStereotype(name).oclIsUndefined();

-- Matched rule
rule CodeList2Enumeration {
  -- Source pattern
  from
    s: UML2!"uml::Class" in IN (
      s.oclIsTypeOf(UML2!"uml::Class") and s.hasStereotype('thecustomprofile::codeList')
    )
  -- Target pattern
  to
    t: UML2!"uml::Enumeration" (
```

```
        name <- s.name,
        ownedLiteral <- s.ownedAttribute -> collect(e | thisModule.Property2EnumerationLiteral(e))
      )
    -- Action block
    do {
      t.applyStereotype('CodeList'.stereotype());
    }
}

-- Lazy rule
lazy rule Property2EnumerationLiteral {
    -- Source pattern
    from
      s: UML2!"uml::Property"
    -- Target pattern
    to
      t: UML2!"uml::EnumerationLiteral" (
        name <- s.name + '=' + s.defaultValue.value
      )
}
```

The keyword *create* denotes the target model to be created, the keyword *from* denotes the source models to be transformed. The models are specified as attributes followed by their type, i. e. the metamodel they are instances of. In the example in listing 3.1, the target model *OUT* is to be created from the source models *IN*, *STANDARDPROFILE* and *COREPROFILE*. All models are instances of the UML 2 metamodel. The latter two models represent UML profiles which are applied to the source and target models, respectively; they have to be provided to the ATL transformation as separate source models.

Furthermore, each transformation module defines transformation rules; two types of transformation rules exist in ATL, *matched rules* and *called rules*. Matched rules are mainly declarative and are used to match model elements from the source model, and to create from these matched model elements the specified model elements in the target model. Called rules, in contrast, can be used to create target model elements using imperative code; they are only executed when called from an imperative code block. Imperative blocks can be defined within the action block of matched rules (see below) or within the body of called rules. Rules can also be marked as abstract and then be extended by other rules.

Matched rules are classified into *standard rules*, *lazy rules* and *unique lazy rules*. Standard rules are executed automatically when the ATL transformation definition is processed, whereas lazy rules and unique lazy rules are only executed when called by another matched rule. Above that, unique lazy rules differ from lazy rules in that they are only executed once and, thus, always deliver the same created target element, even when called repeatedly. The ATL transformation in listing 3.1 contains two matched rules, the standard rule *CodeList2Enumeration* as well as the lazy rule *Property2EnumerationLiteral*. The rules transform model elements which are instances of the UML metaclass *Class* into model elements which are instances of the UML metaclass *Enumeration*. The XMI representation of a corresponding source model element to be matched is displayed in listing 5.1, page 78, the XMI representation of the target model element generated after applying the ATL transformation is shown in listing 5.2, page 82.

Every matched rule consists of a *source pattern* and a *target pattern*. The source pattern defines the elements of the source model to be matched. The pattern starts with the declaration of a source variable whose type is the type of the source model elements to be matched, and is optionally followed by a *guard* in the form of a boolean expression which can be used to filter the source model elements. The source pattern is indicated by the keyword *from*. In the example in listing 3.1, a source variable *s*

of the UML type *Class* is declared, thus, only source model elements which are instances of the UML metaclass *Class* are matched. The keyword *in* is used to specify that only model elements from the source model *IN* are to be considered; furthermore, an additional guard restricts the matching to those UML class elements the stereotype «codeList» is applied to.

The target pattern defines the target model elements to be generated; it is denoted by the keyword *to*. This pattern starts with the declaration of one or more variables which are of the type of the target model elements, and is followed by one or several *bindings* which refer to the variables declared beforehand and which specify how the features of these types are to be initialised. A feature of a type can be an attribute, a reference or an association end. A binding has always the form *featureName <- exp*, the left part defining the feature to be initialised, the right part specifying the expression to be applied for initialisation. In the example in listing 3.1, a target variable *t* of the UML type *Enumeration* is declared; this means that target model elements which are instances of the UML metaclass *Enumeration* are to be generated. The features of the UML metaclass *Enumeration* to be initialised are the attribute *name* and the association *ownedLiteral*. They are initialised with values from the attribute *name* and the association *ownedAttribute* of the UML metaclass *Class*, respectively.[3] Since the association *ownedLiteral* is of the type *EnumerationLiteral* and the association *ownedAttribute* is of the type *Property*, both of them exhibiting their own features, the lazy rule *Property2EnumerationLiteral* needs to be called. This rule specifies that the attribute *name* of the UML metaclass *EnumerationLiteral* is to be initialised with the values from the attribute *name* and the association *defaultValue*[4], both being features of the UML metaclass *Property*, in the form *name=value* (cf. section 7.3.1, page 144).

A matched rule can optionally contain a third section, the *action block*, which starts with the keyword *do*. Within this section, imperative statements can be defined. These statements are executed after the target model elements specified in the *to* section have been initialised. This section can, for instance, be used to modify previously initialised features or to initialise features which have not been initialised in the *to* section. In listing 3.1 the rule *CodeList2Enumeration* uses the *do* section to apply the stereotype «CodeList» to each model element initialised by the rule.

In addition to rules, an ATL transformation can also contain *helpers* which are methods defined by the user. In the example in listing 3.1 the helper *hasStereotype* is specified. It takes a stereotype name as parameter and checks if the stereotype is applied to the source model elements for which the helper is called. Furthermore, also *attributes* can be defined by the user.

The other two kinds of units provided by ATL are queries and libraries. *Queries* allow for querying source models; the result of such a query is always of a primitive data type, i. e. a Boolean, Integer, Real or String type. *Libraries* can be used to define reusable ATL code, such as rules and helpers, externally and import them into another ATL unit as required. Also in listing 3.1 a library is imported, called *UML2*, which provides additional helpers.

Furthermore, the transformation module *UML2Copy* (Wagelaar 2010) is imported. This module provides transformation rules for every metaclass in the UML 2 metamodel by simply copying the source model elements into corresponding target model elements. By using the UML2Copy module in addition to the AAA2Core module defined in listing 3.1, a technique referred to as *module superimposition* (Wagelaar 2008) is applied. Module superimposition means that several transformation modules are superimposed on top of each other and are executed as one unified

[3] All UML metaclasses inherit the *name* attribute from the UML metaclass *NamedElement*. For the definition of the associations *ownedAttribute* and *ownedLiteral* please refer to figures A.2 and A.3 in appendix A, page 171, respectively.

[4] The association *defaultValue* does not contain the required value itself. The association is of the type *ValueSpecification* which contains the attribute *value*; this attribute holds the actual value which is why the ATL expression needs to be written as *s.defaultValue.value* and not simply as *s.defaultValue*.

transformation module. In this way, a base module can be extended by providing additional rules as part of a superimposing module, which holds for the rules defined in the example in listing 3.1. In addition, by naming rules in the superimposing module identically to existing rules in the base module, the existing rules can be overridden; thus, the model elements affected by the superimposed rules in listing 3.1 are not just copied from the source to the target model but transformed in the defined way.

3.4.2 UMLT – A transformation language for information integration

In the context of the research project mdWFS (cf. section 4.3.3, page 67) the transformation language UMLT (UML Transformations) has been developed to meet the needs of creating transformation definitions for performing information integration tasks between source and target models at the conceptual schema level (Staub 2007; Staub et al. 2008; Staub 2009; Donaubauer, Kutzner et al. 2010; Fichtinger 2011). UMLT represents a first-class extension (cf. section 2.4.1, page 19) of the UML specification as it extends the UML 2 metamodel, in particular metaclasses related to UML 2 Activities, by additional metaclasses, allowing in this way for specifying transformation definitions using UML 2 activity diagrams and, thus, based on a graphical syntax. In addition, a textual syntax in the form of a Human-Usable Textual Notation (HUTN) was developed for UMLT which is derived from the Swiss modelling language INTERLIS (KOGIS 2006). The HUTN syntax makes it easier for users to define UMLT transformation definitions. Furthermore, UMLT diagrams can also be represented in the form of the transfer format XMI which helps making the transformation definitions machine-interpretable and allows for providing them to transformation tools to execute the transformations defined between source and target models. To automate the step between the definition of UMLT diagrams and their representation in the XMI format, a UMLT editor was developed by TUM (Elfouly and Kutzner 2013) which allows, on the one hand, for drawing UMLT diagrams tool-based and, on the other hand, for exporting these UMLT diagrams afterwards automatically as XMI documents. UMLT is a hybrid transformation language, it combines the imperative paradigm of the UML activity diagram with the declarative paradigm used for the UMLT extension.

Figure 3.11 displays the metaclasses introduced by UMLT; the white-coloured metaclasses are part of the UML 2 metamodel, the grey-coloured metaclasses belong to the UMLT metamodel. The most atomic component of a UMLT transformation definition is the *AssignmentDefinition* which is used to assign values to attributes of the target model. Values can for instance be constants, attribute values from the source model, geometries or values calculated by functions (e. g. concatenated attribute values using a *StringConcatenator* function, values selected from a value map using a *ValueMapper* function or converted geometry types such as a conversion of the type *GM_Surface* to the type *GM_Line* using a *PolygonToLineConverter* function) (cf. appendix D.4, page 206, for an overview of currently implemented UMLT functions).

Assignment definitions are grouped within *MappingRules* which represent the actual transformation rules between source and target objects. Mapping rules, in turn, are grouped within *TransformationActions*. In figure 3.12 an example transformation action containing a mapping rule and several assignment definitions is displayed. A transformation action extends the UML 2 metaclass *Action*, which is defined in the UML specification as a '[...] fundamental unit of executable functionality. The execution of an action represents some transformation or processing in the modeled system [...]' (OMG 2011c). Furthermore, it extends the UML 2 metaclass *OpaqueAction*, which is an 'action with implementation-specific semantics' (OMG 2011c). Thus, each transformation action represents an executable unit of transformation whose semantics is defined by the individual mappings within each transformation action. The source objects flowing into a transformation action are specified using

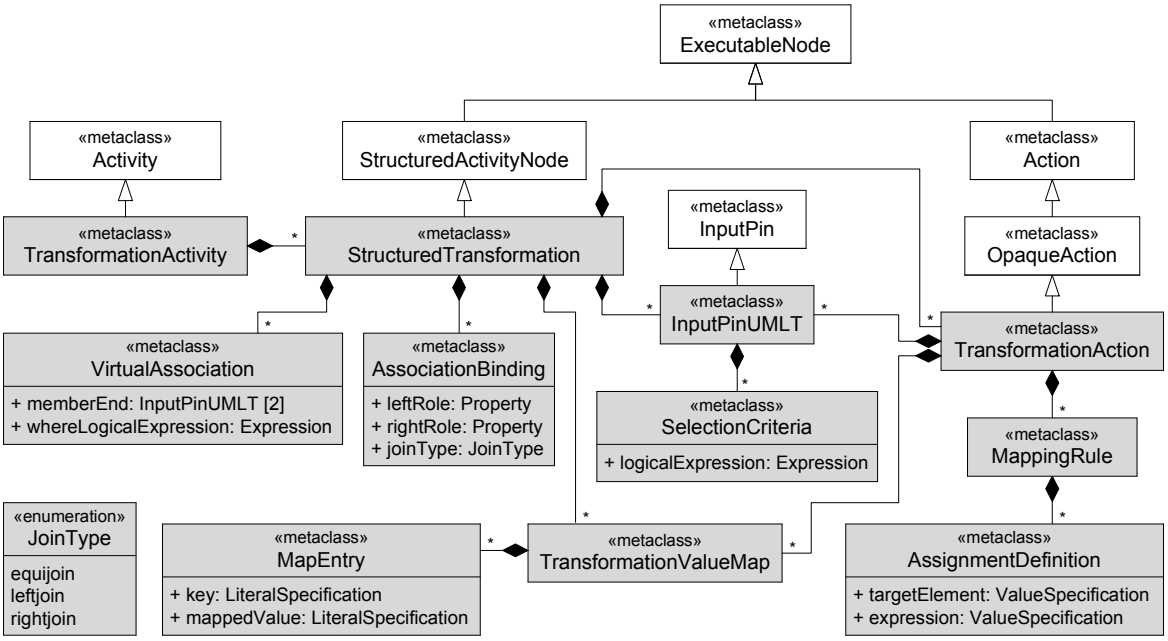


Figure 3.11: Metamodel of the transformation language UMLT; white = metaclasses of UML 2, grey = metaclasses of UMLT. (Staub 2007, modified to reflect all metaclasses defined in the UMLT metamodel and all modifications applied to the UMLT metamodel after its initial publication)

InputPinUMLTs. An input pin UMLT extends the UML 2 *InputPin* by the possibility of applying filters to permit only specific source objects to be processed by the transformation action. These filters are defined by means of *SelectionCriteria*.

Transformation actions are further structured by grouping them within *StructuredTransformations* which, in turn, are grouped within a *TransformationActivity*. A transformation activity extends the UML 2 metaclass *Activity*, which is 'the specification of parameterized behavior as the coordinated sequencing of subordinate units whose individual elements are actions' (OMG 2011c), whereas a structured transformation extends the UML 2 metaclass *StructuredActivityNode*, which 'represents a structured portion of the activity that is not shared with any other structured node, except for nesting' (OMG 2011c). Thus, the transformation activity specifies the complete transformation definition and consists of structured transformations as self-contained action units which are executed in a coordinated sequence.

Another concept introduced by UMLT is the *VirtualAssociation*. A virtual association can be used to associate input objects virtually for the duration of a transformation process in cases where these input objects are not associated to each other explicitly, but exhibit an implicit relationship which needs to be utilised to be able to fully execute the transformation. An example for such implicit relationships are topological relationships, which often are used in geospatial analyses. A virtual association is always evaluated at run-time. A virtual association is defined between attributes from the input objects by specifying a logical expression as value of the attribute *whereLogicalExpression*. A logical expression can be applied to non-geometric and geometric attributes. For instance, to determine all administrative units touching certain administrative boundaries, the objects from a class *AdministrativeUnit* and the objects from a class *AdministrativeBoundary* could be associated using a join criterion *touches*. This

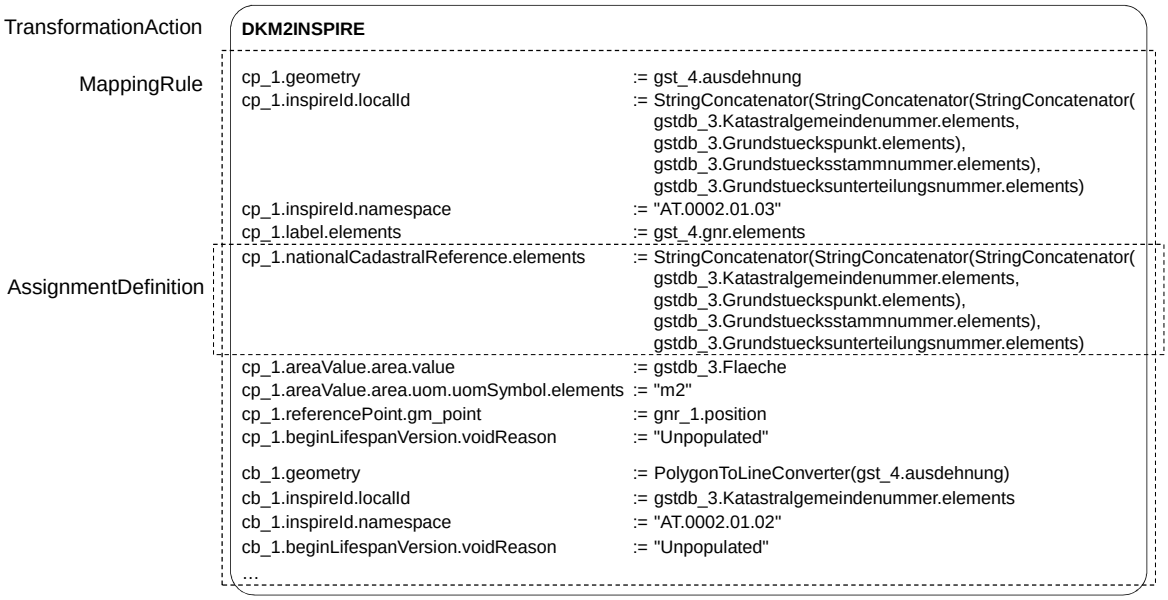


Figure 3.12: Example UMLT transformation action containing one mapping rule and several assignment definitions

is displayed in figure 3.13 for corresponding classes from the German AAA reference model (the attribute *kg* denotes the administrative units, the attribute *gg* refers to the administrative boundaries).

The metaclasses *TransformationValueMap* and *MapEntry* can be used to define value maps consisting of *keys* and *mapped values* for attributes from those source and target objects whose values are defined by enumerations or code lists. An example is given in figure 3.14. The keys denote values used in the German AAA reference model for classifying administrative boundaries, whereas the mapped values represent the values used by the INSPIRE theme Administrative Units for categorising administrative boundaries. Furthermore, by means of the metaclass *AssociationBinding*, the UMLT metamodel also allows for specifying how associated objects and association roles of source objects are treated during the transformation.

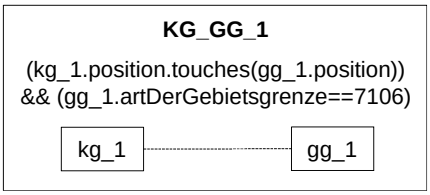


Figure 3.13: Example UMLT virtual association

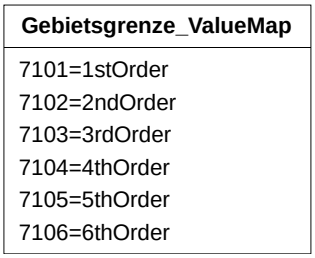


Figure 3.14: Example UMLT value map

A complete UMLT transformation requires more elements than those defined above. These other elements, however, can directly be used from the UML 2 metamodel and are, amongst others, the metaclass *OuputPin* for defining the objects flowing out of a transformation action or structured transformation as well as the metaclasses *InitialNode* and *ActivityFinalNode* for defining a control flow, i. e. the sequence in which the individual transformation actions are processed.

Figure 3.15 exemplifies the use of the different UMLT language elements by defining a transformation definition from the Austrian DKM data model to the INSPIRE theme Cadastral Parcels. The figure displays the overall transformation definition specified by a transformation activity which consists of one structured transformation. Within the structured transformation several virtual associations are defined. Three different input pins define the source objects flowing into the structured transformation, and four different output pins specify the objects resulting from the transformation process. Inside the structured transformation, three transformation actions exist which define the individual mappings between the source and target objects denoted by the input and output pins attached to these transformation actions.

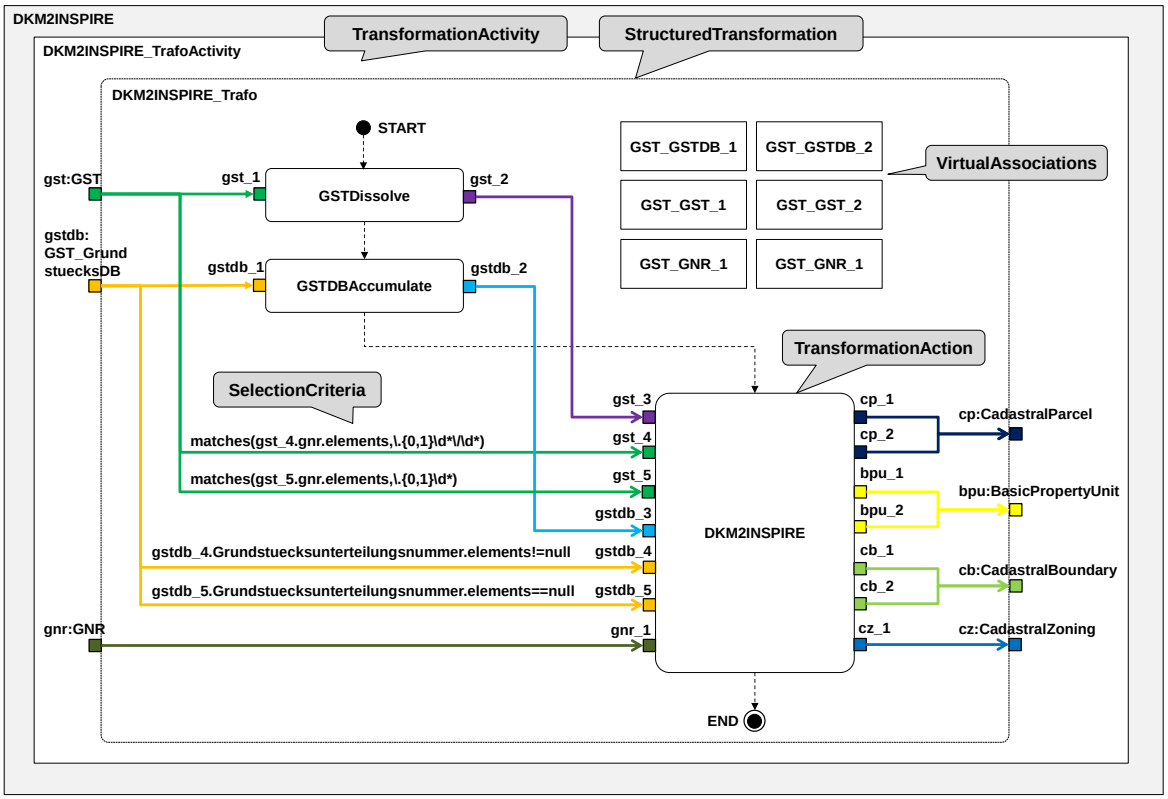


Figure 3.15: Example transformation definition created using the transformation language UMLT

The detailed mappings for the transformation action *DKM2INSPIRE* are in part shown in figure 3.12. Within this transformation action, amongst others, cadastral parcel and cadastral boundary objects are created (represented by the output pin names cp_1 and cb_1, respectively), by assigning values from the attributes of the source objects gst_4, gstdb_3 and gnr_1 to the attributes of the target objects. The first row, for instance, assigns the value from the attribute *ausdehnung* (extent) to the attribute *geometry*, whereas in the second row values from several source attributes need to be concatenated first using the UMLT function *StringConcatenator*, before the complete value can be assigned to the attribute *localId*. Values do not always need to be taken from source objects; also constant values can be assigned to attributes, as is shown in the third row for the attribute *namespace*.

The number of input pins and the number of output pins do not have to be identical, neither for structured transformations, nor for individual transformation actions. Some objects can be mapped

directly from one input pin to one output pin; sometimes, however, objects from several input pins are required to be able to create objects for one output pin and vice versa. In figure 3.12, for instance, values from all three source objects are used to create cadastral parcel objects. Furthermore, on the source side of figure 3.15 only cadastral parcel objects are flowing into the transformation (input pin gst:GST), whereas on the target side cadastral parcel objects and cadastral boundary objects (output pins cp:CadastralParcel and cb:CadastralBoundary, respectively) are flowing out of the transformation. Besides mapping the source cadastral parcel objects to the target cadastral parcel objects, also the cadastral boundary objects can be created from these source cadastral parcel objects by deriving linear geometries from their polygon geometries using the UMLT function *PolygonToLineConverter* within the transformation action *DKM2INSPIRE* (cf. figure 3.12).

3.4.3 Other relevant transformation languages in the geospatial domain

Besides the two transformation languages presented above, many other transformation languages exist. Four of them are introduced briefly in the following since they are of importance in the context of transforming geospatial data as well.

As will be pointed out in section 4.3, page 63, in the analysis of state-of-the-art transformation approaches of geospatial data, transformation definitions are today still defined predominantly at the data format (schema) level and not at the conceptual schema level. This might, on the one hand, be due to the fact that not all available geospatial data are already defined conceptually by means of a conceptual schema language. On the other hand, it might also be due to the fact that in the geospatial domain in particular the XML-based transfer format GML is widely-used and, thus, transformations are conveniently defined at the level of GML documents or GML application schemas, i. e. at the data format (schema) level – often by means of the transformation language *Extensible Stylesheet Language Transformations* (XSLT). XSLT is a W3C standard for transforming XML documents (Kay 2007). By means of XSLT, transformation definitions can be defined at the data format level and since XSLT version 2.0 to a certain extent also at the data format schema level, which, however, requires the use of a schema-aware XSLT processor for executing the transformation definitions in a correct way (Kay 2007). XSLT is a declarative language. XSLT is targeted at the XML paradigm; therefore, its usage with data formats conforming to other modelling paradigms is possible, but convenient to a limited extent only. This also holds for the use of XSLT for transforming GML application schemas which comply to the OO paradigm. GML is based on XML only in so far as for the definition of GML application schemas the XML syntax is used. As regards the semantics, however, the GML specification requires that '[t]o ensure proper use of the conceptual modelling framework of the ISO 19100 series of International Standards, all application schemas are expected to be modelled in accordance with the General Feature Model as specified in ISO 19109' (ISO 2007), which, in turn, corresponds to the OO paradigm (cf. section 2.3.2, page 14) (Kutzner and Eisenhut 2010).

Another W3C standard is the *Rule Interchange Format* (RIF) (Kifer and Boley 2013). RIF is a formal declarative language for describing rules in the context of the Semantic Web and for exchanging them between heterogeneous rule systems in the form of XML documents. RIF consists of three dialects, RIF-Core which provides a core syntax and semantics, RIF Basic Logic Dialect (RIF-BLD) for defining logic rules and RIF Production Rule Dialect (RIF-PRD) for defining production rules, the latter two extending the syntax and semantics of RIF-Core. In particular RIF-PRD is recommended in the Technical Guidance for implementing INSPIRE transformation services as a suitable language for defining transformation definitions (Howard et al. 2010) (cf. section 4.3.1, page 63). The RIF format is XML-based, however, similar to GML, only in so far as it uses the syntax of XML. As regards the

semantics, it corresponds to the RDF paradigm; therefore, it is not considered suitable for specifying transformation definitions at the conceptual schema level between UML models which correspond to the OO paradigm (Kutzner and Eisenhut 2010). The RDF modelling paradigm is a W3C Semantic Web standard of its own. RDF is commonly expressed using XML syntax. Besides, RIF is closely related to the Web Ontology Language (OWL), another W3C Semantic Web standard. OWL is a formal language for defining ontologies and uses, amongst others, RDF/XML syntax for representing ontologies in the form of a transfer format.

The Ontology Mapping Language (OML) is a declarative language which can be used to define mappings, called alignments, between entities of different ontologies. OML was originally developed by the Ontology Management Working Group (OMWG). Since then, it has undergone redesign and reimplementation as well as a renaming into the Expressive and Declarative Ontology Alignment Language (EDOAL) (Euzenat 2015). EDOAL can be used independently from specific ontology languages. According to (Reitz, Schäffler et al. 2010), one reason for having chosen OML in the HUMBOLDT project (cf. section 4.3.1, page 63) was that it cannot only be used with ontologies, but also with modelling languages such as UML and with data format schemas, such as GML application schemas or database schemas. However, also EDOAL expresses alignments between ontologies using RDF/XML syntax and, thus, corresponds to the RDF paradigm just as RIF does, making it not the best candidate for specifying transformation definitions at the conceptual schema level between OO UML models as well. Within the HUMBOLDT project, a profile of OML was developed, called geographic OML (gOML), to be able to represent transformation requirements specific to geospatial data. This includes, for example, the use of OGC Common Query Language (CQL) expressions to allow for defining (spatial) filters on the geospatial data to be transformed. Using OML, the complete transformation definition is represented by an *Alignment*; the individual transformation rules of the alignment are referred to as *Cells*, each cell defining a transformation rule between two *Entities*, one source and one target entity, which can be classes, instances, relations or properties. An alignment, furthermore, references the schemas being mapped and specifies for each schema the formalism, i. e. the language used for defining the schema (Reitz, Vries et al. 2009).

In contrast to the transformation languages XSLT, RIF and OML, which serve in conducting information integration, *MOF 2.0 Query/View/Transformation* (QVT) (OMG 2011a) represents a set of formal transformation languages which are applied in defining model transformations. The QVT specification is published by the OMG and defines three different languages, QVT Core, QVT Relations and QVT Operational Mappings. The Core and Relations languages are declarative and allow for defining uni- and bidirectional transformations, whereas the Operational Mappings language is imperative and only supports unidirectional transformations. The Operational Mappings language, in addition, extends OCL by imperative expressions. The Relations and Operational Mappings languages can be used jointly, applying a hybrid approach in this way. The Relations language provides a textual and a graphical syntax, the Core and the Operational Mappings languages only offer a textual syntax. QVT is based on MOF and corresponds to the OO paradigm. QVT is used within the MDA architecture (cf. section 3.1.2, page 29) to define horizontal transformations (e. g. PIM $\rightarrow$ PIM) as well as vertical transformations (e. g. PIM $\rightarrow$ PSM). This requires the metamodels of the source and target models to be based on MOF, which holds, for instance, for the UML metamodel and for UML profiles compliant to the UML profile definition of the OMG; thus, QVT is suitable for defining transformations between UML models.

4 Modelling and transformation of geospatial data in the geospatial domain today

After having presented the theoretical background on geospatial data modelling and model-driven transformation of geospatial data, this chapter focuses on the current situation of developing geospatial data models and transforming geospatial data as encountered by the author of this thesis in the geospatial domain today. The first part of this chapter presents various ways of how geospatial data models are currently used and the most predominant problems arising therefrom. The second part provides an overview of the various transformation approaches applied in academia and professional practice.

4.1 State-of-the-art use of conceptual models in the geospatial domain

When dealing with models, it can be noticed that models are used for pursuing different goals; however, depending on their intended use, the models have to meet specific requirements. This section discusses three different categories conceptual models in geoinformation can be classified into (Kutzner and Eisenhut 2010; Kutzner and Donaubauer 2012).

- *Models for communication purposes*: One common use of models is to support communication between people. Putting ideas discussed between people down into writing, in particular in the form of visual models, can considerably facilitate the communication process and serve in creating a common understanding about certain perceptions between people, especially between people from different backgrounds and professions, such as between software developers and GIS experts or between two GIS experts from different application domains. When using models for communications purposes, they only need to be machine-readable – provided they exist in electronic form; they are not yet required to be machine-interpretable as they are interpreted by humans only. Also those data models, which are widely-used today for modelling geographic information, can be assigned to this category. The primary task of these data models is to describe the content and structure of geospatial data for a specific area of application. An example are the application schemas provided by the INSPIRE data specifications. These models represent a common consensus on the content and structure of the data to be provided by the European member states. The models are provided in visual form as UML class diagrams and are easy to understand by everybody familiar with the concepts of the UML language.
- *Models for software development*: As described in section 3.1, page 27, model transformation can be applied to conceptual models to create from them new conceptual models or programming code. This approach requires the models to be parsable and processable by a transformation tool; thus, the models need to be defined in such a way that they are machine-interpretable. However, not only new models or software can be derived from existing models, but also data (transfer) format schemas; in this case the model transformation is also referred to as encoding (cf. section 3.3.1, page 36). In geoinformation, encoding is, for instance, applied to the application schemas of the

INSPIRE data specifications. The application schemas are defined as UML class diagrams from which GML-based data transfer format schemas are derived automatically using an encoding tool; thus, the INSPIRE data specifications can be classified into this category of models. This category also includes application schemas which cannot directly be transformed into data (transfer) format schemas, but which require the creation of an implementation schema (cf. section 2.3.3, page 16) as intermediate step. Regarding the application schemas considered in this thesis, this holds for the German AAA reference model.

- *Models for controlling run-time systems*: Models belonging to this category are charged with controlling the execution of a system. In the context of model transformation all transformation definitions which are specified using a formal transformation language can be classified into this category, as these transformation definitions represent models themselves. The transformation tool is the run-time system which interprets the transformation definition and automatically executes the transformation in a controlled way based on the transformation rules defined in the transformation definition. To be able to control the execution of the transformation tool, transformation definitions have to be fully machine-interpretable. Examples for this kind of models are the transformation definitions created using the transformation languages ATL (cf. section 3.4.1, page 43) and UMLT (cf. section 3.4.2, page 46) as part of the mdWFS project (cf. section 4.3.3, page 67) and of the project the use case of this thesis is based on (cf. section 7.1, page 139), as well as the transformation languages RIF and OML (cf. section 3.4.3, page 50) used in other transformation projects in the geospatial domain (cf. section 4.3.1, page 63).

4.2 Problems arising from the state-of-the-art use of conceptual models in the geospatial domain – a critical review

Several problems arising from the way conceptual models are currently defined and used in the geospatial domain have been identified when trying to apply the approach of model-driven transformation of geospatial data (cf. section 3.3.2, page 39) to the transformation of geospatial data conforming to the German AAA model, the Austrian DKM model, the Swiss MOpublic model, the Swiss GG25 model and the CityGML model into geospatial data conforming to the European INSPIRE data specifications as well as from the Swiss SwissNames model to the European EGN model[1]. These transformations were carried out as part of those research projects the author of this thesis was involved in and which are listed in section 1.1, page 2.

The problems can be classified into three categories and are presented as follows (in part based on (Kutzner and Eisenhut 2010; Kutzner and Donaubauer 2012)): Problems related to the modelling language used (sections 4.2.1.1 – 4.2.1.4), problems related to the encoding rule used (sections 4.2.2.1 – 4.2.2.2) and problems related to the models defined (sections 4.2.3.1 – 4.2.3.2).

4.2.1 Problems related to the modelling language used

4.2.1.1 Semantic modification of the UML specification

Some of the above mentioned UML models exhibit a semantic modification of the UML specification, which means they are not fully conforming to the UML specification any more and, thus, are not completely machine-interpretable. Semantic modification of the UML specification applies, for

[1] Please refer to the list of acronyms for the meaning of the acronyms denoting the data specifications.

instance, to the stereotypes «CodeList» and «Union» defined in the standard ISO/TS 19103 (ISO 2005c) (cf. section 5.1, page 76). According to the UML profile definition, stereotypes are only allowed to extend existing UML metaclasses in such a way that the concepts defined by the UML metamodel remain unchanged (cf. section 2.4.1, page 19). The stereotypes «CodeList» and «Union», however, cause a semantic modification of the UML metaclass *Class* which leads to a first-class extension of the UML metamodel itself, to the effect that the standard ISO/TS 19103 defines a new modelling language, which uses the syntax of UML, but exhibits a semantics of its own. The same problem applies to all UML models which are based on the standard ISO/TS 19103, in particular the German AAA reference model and the INSPIRE data specifications, as they use these semantically modified concepts as well. Another semantic modification of the UML specification is introduced by the INSPIRE data specifications, by defining the stereotype «voidable» for managing those cases, in which data providers are not able to provide values for certain attributes.

When processing these UML models for purposes beyond communication using standard UML tools, problems may occur, since UML models based on a modified UML metamodel might not be interpretable any more in a semantically correct way (cf. section 2.4.1, page 19). This problem is of utmost importance, which is why chapter 5, page 75, discusses the above mentioned stereotypes in detail (cf. sections 5.1.1.1, page 77, and 5.1.2.2, page 83, for the stereotypes «CodeList» and «Union», respectively, and section 5.4.1.2, page 102, for the stereotype «voidable») and provides solutions for how to define the stereotypes semantically compliant to the UML profile definition.

Another example for semantic modification of the UML specification is the Application Domain Extension (ADE) mechanism introduced by the CityGML specification (Gröger et al. 2012). This mechanism allows, on the one hand, for extending CityGML with new feature types and, on the other hand, for enriching existing CityGML feature types with additional application-specific properties. These new properties are modelled as UML subclasses of existing UML classes within a separate UML package and, thus, within a separate namespace. The new UML subclasses are assigned the stereotype «ADEElement» and are named identically to the UML superclasses they extend. Furthermore, the generalisation relationships between the existing UML classes and the new UML subclasses are assigned the stereotype «ADE» (van den Brink et al. 2012). During the encoding, the properties from the UML subclasses are injected into exactly those objects which are instances of the corresponding UML superclasses, applying a so-called *super class strategy*. This approach brings about one decisive advantage: the enrichment of existing CityGML feature types with additional properties can be represented visually at the conceptual schema level in an easily and clearly understandable way, while at the data format level the UML subclasses are suppressed from being encoded in the CityGML instance document as additional, independent objects with identifiers of their own and, thus, with distinct identities. In this way, each object is represented exactly once in the CityGML instance document, containing at the same time standard CityGML properties belonging to the object-specific namespace and additional, application-specific properties belonging to ADE-specific namespaces. Furthermore, standard CityGML objects can be enriched with application-specific properties from several ADEs at the same time, turning the CityGML model into an information hub in this way, which allows for coupling applications from different disciplines (Kolbe 2009).

Figure 4.1 shows an example of using the ADE mechanism. The UML class *AbstractBuilding* with the stereotype «ADEElement» is such a UML subclass which extends the equally named CityGML superclass by properties required for noise immission simulations. The corresponding CityGML instance is shown in listing 4.1. The properties *buildingReflection* and *buildingReflectionCorrection* which have been part of the UML subclass in figure 4.1 are now part of that building object which is an instance of the UML superclass. The only difference is that these properties belong to the

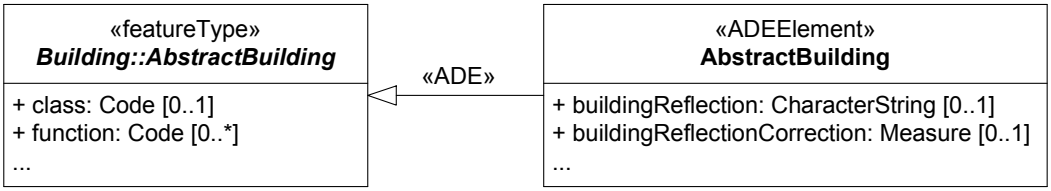


Figure 4.1: Excerpt from the CityGML Noise ADE (Gröger et al. 2012) defining additional properties as UML subclass of the UML class *AbstractBuilding*

ADE namespace (namespace prefix *noise*), in contrast to the existing CityGML properties *class* and *function*, which belong to the namespace of the CityGML Building module (namespace prefix *bldg*), as does the building object itself. Listing 4.2, in contrast, demonstrates what the CityGML instance would look like without the super-class strategy. The single building object from listing 4.1 is now represented in the form of two distinct building objects; building object *building1*, which is an instance of the UML superclass within the object-specific namespace and which contains only the standard CityGML properties, and building object *building2*, which is an instance of the UML subclass within the ADE-specific namespace and which contains not only the ADE-specific properties, but also inherits the standard CityGML properties from the UML superclass. Although only one real-world building is to be described by the CityGML instance, this real-world building is represented by two separate building objects. According to the OO paradigm, each object has its own identity; since the identifiers of the two building objects differ, they denote, in fact, two different real-world buildings – a circumstance, which can be avoided by using the super-class strategy.

Listing 4.1: CityGML encoding of figure 4.1 applying the ADE super-class strategy (Gröger et al. 2012)

```
<bldg:Building gml:id="building1">
  <bldg:class>1000</bldg:class>
  <bldg:function>2130</bldg:function>
  ...
  <noise:buildingReflection>Fassade</noise:buildingReflection>
  <noise:buildingReflectionCorrection uom="dB">3.23</noise:buildingReflectionCorrection>
  ...
</bldg:Building>
```

Listing 4.2: CityGML encoding of figure 4.1 without applying the ADE super-class strategy

```
<bldg:Building gml:id="building1">
  <bldg:class>1000</bldg:class>
  <bldg:function>2130</bldg:function>
  ...
</bldg:Building>

<noise:Building gml:id="building2">
  <bldg:class>1000</bldg:class>
  <bldg:function>2130</bldg:function>
    ...
  <noise:buildingReflection>Fassade</noise:buildingReflection>
  <noise:buildingReflectionCorrection uom="dB">3.23</noise:buildingReflectionCorrection>
    ...
</noise:Building>
```

However, one important disadvantage of this approach is that, similar to the stereotypes «CodeList» and «Union» mentioned above, also the stereotype «ADEElement» causes a semantic modification of the UML metaclass *Class*, to the effect that the semantics of the stereotype is understandable visually, but is not machine-interpretable and, thus, e. g. the creation of CityGML XML Schema documents using existing UML tools is not possible straight-away. Another disadvantage is that 'it is not possible to create an instance diagram where an object instance is shown combining properties from different ADE's. The only way to show instance data is on the XML level' (van den Brink et al. 2012). Furthermore, it should be noted that the stereotypes are not provided as part of a UML profile as is required by the UML 2 specification (cf. section 2.4.1, page 19); in fact, these stereotypes actually require the definition of a specific CityGML UML profile (cf. section 5.5.4, page 111).

4.2.1.2 Use of different UML profiles

In general, each community may define a UML profile of its own and apply it to the UML models defined within this community, to the effect that a transformation process involving UML models from different communities might possibly face many different UML profiles. This problem also occurs in the transformation conducted from the German AAA reference model and the Swiss MOpublic model to the INSPIRE data specifications. Apart from being based on the standard ISO/TS 19103, the German AAA reference model and the INSPIRE data specifications define specific UML profiles which include additional stereotypes (cf. sections 5.5.1, page 108, and 5.4, page 100, respectively). Furthermore, also INTERLIS provides a UML profile of its own (cf. section 5.5.2, page 109).

The definition of new stereotypes does not seem difficult at first glance, which is why the possibility of adapting UML to the needs of a specific community may appear very convenient. However, the definition of a new UML profile might be accompanied by consequences one has to be aware of beforehand. One consequence is the semantic modification of the UML specification described above in section 4.2.1.1. Each community which defines stereotypes in this way, at the same time also defines a new modelling language with community-specific semantics which, in turn, poses a problem for processing the models by means of existing UML tools. Furthermore, community-specific semantics, be it semantically compliant to the UML specification or not, can result in difficulties regarding the transformation between UML models in a cross-community environment. For this reason, this problem will be addressed in detail as part of the following chapters.

4.2.1.3 Use of different UML versions

The standard ISO/TS 19103 is based on UML version 1.4.2 and, thus, each UML model which is defined compliant to the standard ISO/TS 19103 shall make use of UML version 1.4.2 as well (ISO 2005c); this holds e. g. for the German AAA reference model (cf. section 5.5.1, page 108). An exception to this provide the INSPIRE data specifications which are based on the standard ISO/TS 19103 as well, but for which explicitly the use of UML version 2.1 is specified (JRC 2014a). Besides, since the standard ISO/TS 19103 is currently under revision, UML models which will based on the revised standard might in future use UML version 2.4.1 (ISO 2013a) (cf. section 5.1.3, page 85). Furthermore, other UML models might have to be processed which are not based on the standard ISO/TS 19103; these UML models are defined according to a specific UML version, too.

In particular the use of UML models based on UML 1 together with UML models based on UML 2 can lead to difficulties in their combined processing. One problem which can appear refers to the different way of defining and using stereotypes in UML 1 and UML 2. In UML 2 all stereotypes have

to be defined in the context of a UML profile and tag definitions can only exist as properties of a stereotype. Furthermore, several stereotypes can be applied to a UML model element. UML 1, in contrast, allows stereotypes to be defined and applied to UML model elements without having to specify them formally as part of a UML profile beforehand; however, only one stereotype can be applied to a UML model element. Also tag definitions can be added to UML model elements irrespective of whether a stereotype has been applied to the UML model element or not (cf. section 2.4.1, page 19).

Another example for how the combined use of UML versions 1 and 2 can lead to problems, is provided by the data types *Boolean* and *Integer*. UML 1 does not prescribe the use of specific data types and also does not offer predefined data types. However, the standard ISO/TS 19103, which is based on UML 1.4.2, defines several fundamental data types relevant for geographic information such as *Date* and *Time*, *Integer* and *Real*, *Boolean* and *Length*. UML 2, in contrast, provides four predefined primitive data types (*Integer*, *String*, *UnlimitedNatural* and *Boolean*) and, thus, leads to the existence of a second *Boolean* and *Integer* data type. When transforming a UML model which has been defined compliant to the standard ISO/TS 19103 and, thus, is based on UML 1, into a UML model based on UML 2, compatibility problems might occur. It has to be clarified prior to the transformation process how the data types *Boolean* and *Integer* are handled in UML 2. This is of particular importance for the data type *Boolean* since UML 2 defines for this type the values *true* and *false*, whereas ISO/TS 19103 states the values with *TRUE = 1* and *FALSE = 0*[2]. Furthermore, when UML models are to be defined compliant to the standard ISO/TS 19103, but shall explicitly make use of UML 2, it needs to be decided prior to starting the modelling process which data types from ISO/TS 19103 and UML 2 are to be employed for modelling.

A further problem which can appear is that in UML 2 some UML metaclasses from which to extend a stereotype have changed. To assign a stereotype to a UML attribute or association, the stereotype needs to extend the UML metaclasses *Attribute* and *AssociationEnd* in UML 1, whereas it needs to extend the UML metaclass *Property* in UML 2.

4.2.1.4 Discrepancy between visual and machine-interpretable representation of UML models

The conceptual UML models applied in the transformations are defined in such a way that they merely represent models for communication purposes (cf. section 4.1, page 53) and, thus, only provide for machine-readability, but not for machine-interpretability, resulting in a discrepancy between the visual and the machine-interpretable representation of the UML models.

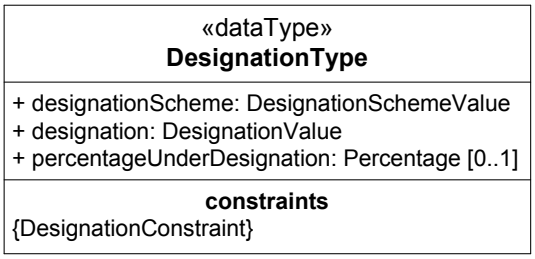


Figure 4.2: Data type *DesignationType* from the application schema INSPIRE Protected Sites Simple (JRC 2014f)

The application schema *INSPIRE Protected Sites Simple* (JRC 2014f) provides an example for this problem. The application schema defines the data type *DesignationType* which contains the attribute *designation* of the type *DesignationValue* (cf. figure 4.2). *DesignationValue*, in turn, is defined as a code list which is associated to several other code lists by means of a generalisation relationship (cf. figure 4.3). From a visual point of view it is

[2]Since UML 2 and ISO/TS 19103 specify the values as enumeration literals, also case sensitivity plays a role here.

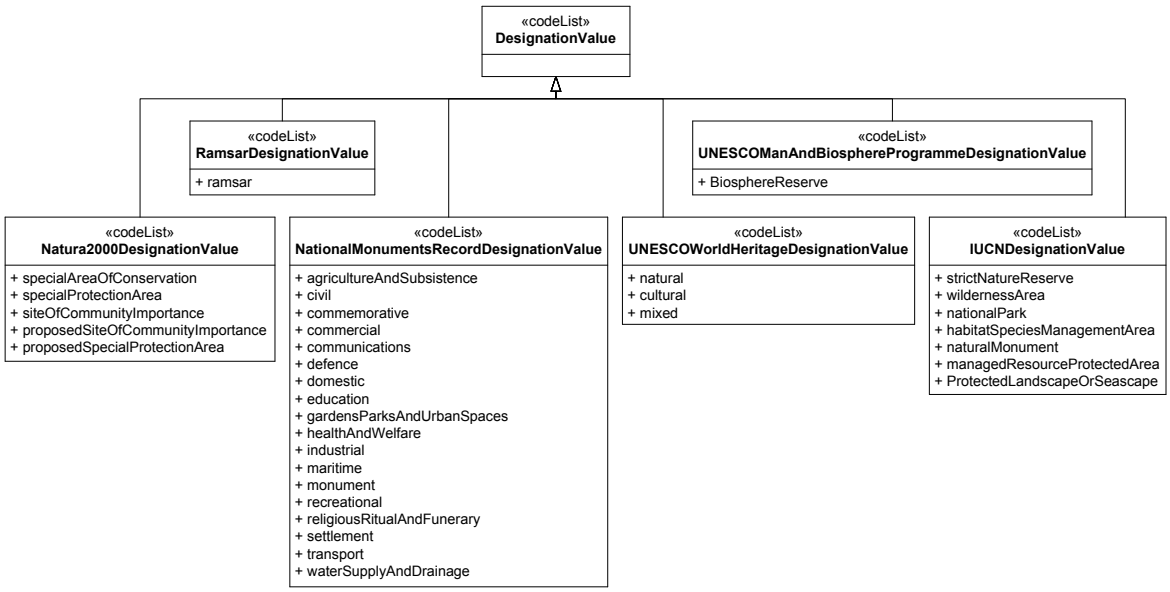


Figure 4.3: Code list *DesignationValue* with specialising code lists from the application schema INSPIRE Protected Sites Simple (JRC 2014f)

evident that the attribute *designation* is to be assigned a value from one of the specialising code lists of *DesignationValue*. However, from a machine-interpretable point of view it is not possible to assign a value to the attribute. According to the definition provided by the UML specification for the UML metamodel concept *Generalization*, only the specialising class knows which base class it references and inherits properties from; the base class does not even know that it is part of a generalisation relationship: 'A generalization relates a specific classifier to a more general classifier, and is owned by the specific classifier' (OMG 2011c). Thus, a generalisation relationship can be interpreted by a standard UML tool only from the specialising class to the base class (bottom-up), but not from the base class to the specialising class (top-down). For this reason, only values from the code list *DesignationValue* itself can be assigned to the attribute *designation* – in this case no value at all, as the code list does not define any values –, but not from one of the specialising code lists.[3]

Another example is the data type *GeographicalName* from the application schema INSPIRE Geographical Names which is displayed in figure 4.4. The data type defines two attributes of the type *CharacterString*. Processing the application schema using a UML tool revealed that two different types of *CharacterString* were assigned to the attributes. On the one hand, the attribute *sourceOfName* was assigned a *CharacterString* type defined in the standard ISO/TS 19103. This type is a complex type, i. e. it does not just represent a value, but it consists of the four attributes *size*, *characterSet*, *maxLength* and *elements*, the latter one holding the actual value of the attribute *sourceOfName*. On the other hand, the attribute *language* was assigned a primitive type provided by Enterprise Architect, the modelling tool used for defining the Protected Sites Simple UML diagram, which is referred to as *EAJava_CharacterString* in the XMI representation of the UML diagram.

[3]Moreover, even the standard ISO 19136 Annex E explicitly does not allow for specifying generalisation relationships between code lists. Since the INSPIRE Generic Conceptual Model (cf. section 5.4, page 100) recommends defining INSPIRE application schemas compliant to the standard ISO 19136 E.2.1.1.1-E.2.1.1.4. (JRC 2014a), a second reason holds why the use of a generalisation relationship is inappropriate here.

For being able to process these application schemas in a machine-interpretable way, it is necessary that the UML diagrams present the universe of discourse not only visually in a correct way, but that also the underlying UML model is technically modelled in a correct way. The problems presented here can be solved by exercising more care during the modelling process, which requires primarily the modeller having to bear in mind a correct use of the UML specification and of the employed UML modelling tool when defining UML models. Only then UML models can be used beyond pure visual representation for semantic transformations at the conceptual schema level.

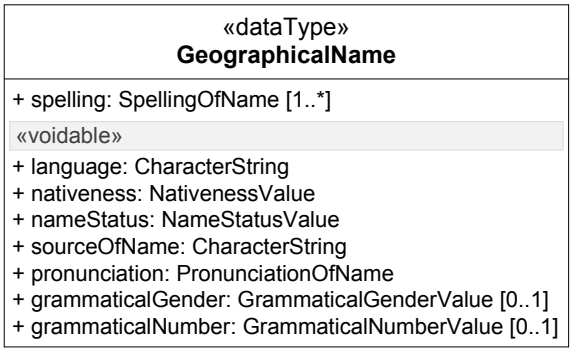


Figure 4.4: Data type *GeographicalName* from the application schema INSPIRE Geographical Names (JRC 2014c)

4.2.2 Problems related to the encoding rule used

4.2.2.1 Documentation of encoding rules

To be able to derive a data transfer format schema from an application schema, an appropriate encoding rule is required (cf. section 3.3.1, page 36). The encoding rules of the German AAA reference model and of the INSPIRE data specifications are documented in (AdV 2009) and (JRC 2014b), respectively, in an informal way using natural language. To be able to execute an encoding automatically, the encoding rule is not required to be present in a formal way based on a formal language. An encoding rule rather needs to be described such precisely using natural language that it can be implemented unambiguously. Otherwise, there is a risk that the encoding rule is interpreted and implemented in different ways by different persons. Unambiguously defined encoding rules ensure that they are executed on every system in exactly the same way without exception.

However, encoding rules might not always be documented. In such cases an encoding rule can be created, nonetheless, by using a reverse engineering approach, i. e. by deducing the encoding rule from the application schema and the corresponding data transfer format schema, provided both are known. As this approach might be error-prone and not be realisable automatically, the transformation rule should in this case rather be defined directly at the data format schema level, which means that only a format-schema-driven transformation can be applied, but not a model-driven transformation (cf. section 3.3.2, page 39). However, to be able to execute the transformation correctly at the data format schema level, the semantics of both, the source and target application schemas, need to be fully present in the corresponding source and target data transfer format schemas.

As example the EuroGeoNames project (EuroGeographics 2015) is mentioned here. The EuroGeoNames project was originally conducted from 2006 to 2009 and aimed at developing a gazetteer web service for providing geographical names. Within this project a conceptual schema for a transnational gazetteer, a database schema and a GML application schema were defined; the corresponding encoding rules, however, were not documented which resulted in some non-understandable encodings[4]. For instance, the UML class *SI_LocationInstance* defined in the conceptual schema was not present in

[4] In 2012 a new gazetteer service was implemented which is based on simplified schemas (Latvala et al. 2013). The problems mentioned in this thesis refer to the original schemas.

the database schema any more, whereas in the GML application schema the abstract UML class *EGN::LocationInstance* became instantiable and UML generalisation relationships between the UML class *EGN::LocationInstance* and its UML subclasses were represented as associations.

4.2.2.2 Documentation of transformation definitions for the derivation of implementation schemas

Sometimes, the application schema is not to be encoded directly as data transfer format schema, but an implementation schema is to be generated first from the application schema using a corresponding transformation definition (cf. section 3.3.1, page 36). For the documentation of transformation definitions the same explanations apply as described above in section 4.2.2.1 for encoding rules; this means, a transformation definition does not need to be defined formally using a formal language, it only needs to be described such precisely by means of a natural language that the implementation schema can be derived unambiguously from the application schema. Furthermore, when the transformation definition is not documented, one should rather work directly with the implementation schema instead of trying to deduce the transformation definition from the application schema and the corresponding implementation schema, provided that the implementation schema completely reflects the semantics of the application schema. Another possibility would be to define the transformation directly at the data format schema level, in case not only the transformation definition is not documented, but also the implementation schema is not provided.

4.2.3 Problems related to the models defined

4.2.3.1 Non-existent application schemas

The use case applied in this thesis (cf. section 7.1, page 139) required the transformation of geospatial base data from the Austrian DKM to the INSPIRE theme Cadastral Parcels. However, at the time the project was conducted, no public application schema existed for the source data. The source data were provided as Shapefile and text file documents whose structure and contents were described in corresponding data transfer interface documents (BEV 2008a; BEV 2008b). To be able to define a mapping at the conceptual schema level despite of a missing application schema, the source application schema needed to be remodelled in a reverse engineering process using UML. The remodelling was based on the data transfer interface documents and, thus, on the concepts of specific platforms, i. e. the Shapefile and the text file platforms, with the result that the UML application schema created reflects the concepts of these platforms.

The MDA approach postulates that a conceptual model is always to be defined in such a way that the concepts used in the model are independent from a specific platform. However, it can be difficult to identify the application-specific concepts unambiguously, once the platform-specific aspects are integrated in a data transfer format. Thus, without any knowledge about the area of application, the probability of being able to remodel application schemas from data transfer formats free of any errors is limited. Similarly, a database model might, for instance, include platform-specific optimisations which cannot be undone without any knowledge about the application-specific concepts forming the basis of the database model.

Furthermore, when remodelling an application schema using UML it needs to be determined which UML version is to be used and whether the UML model is to be defined taking into account a specific UML profile. These decisions should preferably be made in such a way that the problems mentioned in the sections 4.2.1.2, page 57, and 4.2.1.3, page 57, regarding the use of different UML profiles and UML versions are avoided.

4.2.3.2 Representation of spatial attributes

The standard ISO 19109 defines that a feature can contain one or several spatial attributes whose values are geometrical and topological objects from the standard ISO 19107. Two possibilities for how to represent spatial attributes in a UML application schema are mentioned there:

- On the one hand, the spatial attribute can be represented as *UML attribute* of a UML class, the data type of the UML attribute being a spatial type from the standard ISO 19107. Figure 4.5 shows an example from the application schema INSPIRE Buildings. The UML class *Building* defines, amongst others, the UML attribute *geometry3DLoD3* of the complex type *BuildingGeometry3DLoD*. This type, in turn, defines the UML attribute *geometrySolid* whose type is the spatial type *GM_Solid* from the standard ISO 19107. This form of representation corresponds to the concept of *value semantics* available in the OO paradigm which means here that each feature has to be provided with a geometry object of its own.
- On the other hand, the spatial attribute can be represented in the form of a *UML association* between a UML feature class and a geometry type from the standard ISO 19107. Figure 4.6 depicts an example from the CityGML application schema. The UML class *AbstractBuilding* defines, amongst others, the UML association *lod3Solid* whose type is the spatial type *GM_Solid* from the standard ISO 19107. In the OO paradigm, this form of representation corresponds to the concept of *reference semantics*, meaning here that several features can reference, i. e. share, the same geometry object. In this way also topological relationships between different objects can be modelled, as is explicitly provided for in the CityGML specification (Gröger et al. 2012). The example in figure 4.6, for instance, defines that adjacent buildings can share the same wall geometry where they touch each other. Thus, when using reference semantics, it has to be taken into account that changing the geometry value for one object will simultaneously change the value for all objects referencing this geometry.

Although both examples define that a building object can geometrically be represented by a solid geometry, the use of value and reference semantics can result in UML representations whose underlying spatial semantics differs. This can cause problems when transforming between source and target models which differ with respect to this semantic representation. When transforming a model based on reference semantics into a model based on value semantics, shared geometries and topological relationships from the source model cannot be preserved in the target model.

This problem also applies to the transformation of the CityGML application schema to the INSPIRE theme Buildings. The reason for this becomes clear, when implementing the UML model excerpts shown in the figures 4.5 and 4.6 as INSPIRE GML and CityGML instance documents which are displayed in the listings B.1 and B.2, page 176, respectively). In the CityGML document, the shared geometry is represented by the geometry with the identifier *wallSurface4711*. Only the building object *building1* contains this geometry, whereas the building object *building2* references this geometry. In the INSPIRE document, however, the shared geometry from the CityGML document had to be duplicated and to be listed once in the building object *building1* (with identifier *wallSurface4711*) and a second time in the building object *building2* (with identifier *wallSurface0815*). Each of these geometries exhibits an identifier of its own and, thus, represents an independent geometry object of distinct identity. The information that these two buildings actually share the same wall vanished and can only be reconstructed by performing suitable spatial analyses on these two building objects. Also, the topological relationship is not present anymore.

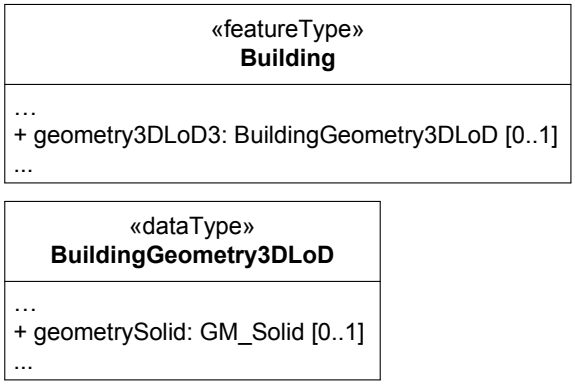


Figure 4.5: Spatial attribute *geometrySolid* represented as UML attribute in the application schema INSPIRE Buildings (JRC 2013)

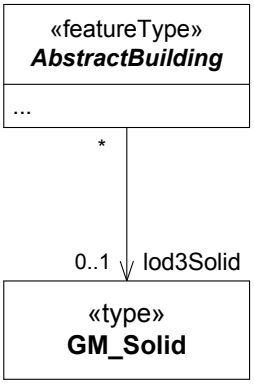


Figure 4.6: Spatial attribute *lod3Solid* represented as UML association in the CityGML application schema (Gröger et al. 2012)

4.3 State-of-the-art transformation approaches of geospatial data in research and industry

Model-driven transformation of geospatial data was identified as a key interoperability issue in many SDI-related initiatives and projects conducted during the last 15 years. Besides, a number of commercial and open-source tools have appeared in this period which allow for transforming geospatial data. The next two sections provide an overview of the most relevant projects and tools which address this topic, focusing on the following viewpoints:

- Was a conceptual modelling language used for defining the source and target data models?
- Which transformation language was used for specifying the transformation definition?
- Was the transformation definition specified at the conceptual schema level, at the data format schema level or at the data format level?
- Which transformation tool was used for executing the transformation?

Afterwards, the mdWFS approach is presented which allows for transforming geospatial data fully compliant to the concept of model-driven information integration at the conceptual schema level. This approach and the outcomes of the corresponding research project provided the basis for the research conducted in the following chapters. The state-of-the-art review is concluded by taking a look at other relevant works from academia tackling the subject of geospatial data modelling and model-driven transformation.

4.3.1 SDI-related research and development projects

GiMoDig (Geospatial Info-Mobility Service by Real-Time Data-Integration and Generalisation) was one of the first EU projects dealing explicitly with the topic of transformation of geospatial data. Aim of the project was the development of a map service which provides harmonised, cross-border topographic maps to mobile users by employing real-time integration and generalisation of heterogeneous source data to allow for displaying maps conveniently on small displays (Afflerbach et al. 2004). Within the project, a common target model was developed in the form of a feature catalogue using natural language. Transformation definitions were specified in tabular form between the national

feature catalogues and the common target model as well as in the form of SQL queries which refer to the national databases (Illert and Afflerbach 2003). Furthermore, a GiMoDig Integration Service was implemented prototypically. This service is capable of transforming GML documents compliant to national GML application schemas into GML documents compliant to a GML application schema based on the common target model. The transformation definitions for this prototype were defined at the data format level using the transformation language XSLT (cf. section 3.4.3, page 50) and were executed using the XSLT processor Xalan (Lehto and Sarjakoski 2004).

The EU project ORCHESTRA (Open Architecture and Spatial Data Infrastructure for Risk Management) aimed at developing a service-oriented architecture for improving the interoperability between different stakeholders involved in risk management in Europe. The geospatial data retrievable through the ORCHESTRA architecture are compliant to an information model framework developed as part of the ORCHESTRA project. Basis of this framework is an ORCHESTRA Meta-Model (OMM) which was defined using UML and which is based on the General Feature Model of the standard ISO 19109 (cf. section 2.3.2, page 14), extending it by additional metaclasses. The metaclasses and rules of the OMM are used for defining ORCHESTRA Application Schemas (OAS); ORCHESTRA Feature Sets (OFS), in turn, are instances of the OAS (Usländer 2007). Thus, ORCHESTRA provides for the possibility of defining target data models at the conceptual schema level using UML. Two types of transformation services were specified within the project, a Schema Mapping Service (SMS) and a Translating Feature Access Service (FAS-X). The transformation definitions can be specified using transformation languages such as XSLT, XQuery or SPARQL. However, the service specifications do not prescribe a certain language to be used; this rather has to be determined by the corresponding implementation specifications (Usländer 2007). A prototypical implementation exists only of the FAS-X, which was realised by means of the software Go Publisher (cf. section 4.3.2, page 66) (Friis-Christensen et al. 2008).

The EU project ESDIN (European Spatial Data Infrastructure with a Best Practice Network) focused on developing best practices for implementing the INSPIRE Directive within the EU Member States. One key aspect was on providing a best practice on transformation approaches most suitable for transforming geospatial data from the individual EU Member States to INSPIRE-compliant geospatial data (Lehto 2011). Within the project, a framework for specifying transformation definitions was developed which proposes a template in the form of a spreadsheet for defining mappings of feature types, attributes and attribute values between a source model and a target model by providing particular mapping expressions (Nissen et al. 2011). The target model, i. e. the INSPIRE data specifications, is defined conceptually using UML; as regards the source models involved in the project, nothing is documented in the public deliverables, and also the framework does not give instructions at which level the mappings are to be defined. However, from the examples given, it can be assumed that within the project the mappings were mainly specified between database schemas on the source side and GML application schemas on the target side and, thus, at the data format schema level. This is supported by the descriptions provided in (Lehto 2011), according to which the transformations were mainly executed between source data stored in databases and target data accessible as INSPIRE-compliant GML documents. Languages and tools employed for defining and executing the transformations were, amongst others, SQL scripts, XSLT, FME (cf. section 4.3.2, page 66) and specifically implemented applications.

Another relevant project, commissioned by the Joint Research Centre (JRC), focused on the development of a Technical Guidance for implementing INSPIRE transformation network services (TNS). As the result of a state-of-the-art analysis of modelling languages, transformation languages and transformation tools conducted at the beginning of the project (Beare, Howard et al. 2010), the

XML-based languages GML and RIF-PRD (cf. section 3.4.3, page 50) were identified and proposed in the Technical Guidance as the most suitable modelling and transformation languages, respectively (Howard et al. 2010). By specifying transformation definitions using RIF, the transformation definitions are interoperable and can be provided to any TNS implementation, where they simply have to be translated into the tool-specific transformation language used internally by the transformation tool of the specific TNS implementation, provided that this can be done unambiguously and that the semantics are preserved. GML was chosen since it is interoperable with RIF; GML application schemas can be imported into RIF documents, and object definitions can, in this way, directly be referenced during the creation of the transformation definitions. Furthermore, GML is – in contrast to the other two considered languages RDF and OWL (cf. section 3.4.3, page 50) – suitable for geospatial data modelling. This means that for being able to use RIF, the transformation definitions need to be defined between source and target GML application schemas and, thus, at the data format schema level. Geospatial data available in other formats first have to be converted to GML. A prototypical implementation of the TNS was realised by means of the software tools HUMBOLDT Alignment Editor (HALE) (cf. section 4.3.2, page 66) and Radius Studio. Spreadsheets describing the transformation definitions were created based on the INSPIRE UML models and the source and target GML application schemas. These transformation definitions were then realised using HALE and were exported as RIF documents (by means of a plug-in specifically developed for HALE). Afterwards, the RIF documents were imported into Radius Studio and converted into the internal rules language of Radius Studio, where the geospatial data were then transformed (Beare, Payne et al. 2010).

The EU project HUMBOLDT aimed at developing an open-source software framework for cross-border data harmonisation in the context of INSPIRE. Harmonisation of heterogeneous geospatial data from the different EU Member States was one of the main issues in the project, which resulted in several tools dealing with modelling and transformation of geospatial data. This includes the HUMBOLDT GeoModel Editor for defining source and target models, HALE for specifying transformation definitions as well as executing transformations and the Conceptual Schema Translation (CST) Service for being able to execute transformations in a web-based environment. Other developments are the modelling language HUMBOLDT Modelling Language (HML) and the transformation language geographic OML (gOML). HML was developed as a DSL, the HUMBOLDT project representing the domain the language focuses on. HML bases itself on conceptual ideas from the ISO 191xx series of geographic information standards and from the modelling languages INTERLIS 2 and UML, but it specifies only those model elements which are required for creating conceptual models within the HUMBOLDT project domain. The HML metamodel was created as an instance of the meta-metamodel Ecore. gOML was developed as a profile of the Ontology Mapping Language (OML) and is used internally by HALE to represent and store the created transformation definitions (please refer to section 3.4.3, page 50, for more information on OML and to section 4.3.2, page 66, for more details on HALE). The GeoModel Editor is intended to be used for defining source and target models based on the modelling language HML. In cases where only source data are available, but no corresponding source model for defining the transformation definition, the editor is to be used for creating the missing source model in a reverse engineering process. Problems which can occur from reverse engineering conceptual models, as mentioned in section 4.2.3.1, page 61, are not considered here. Another intended use of the GeoModel Editor is to import existing models into the editor to enrich them, if required, with information necessary for being able to create adequate transformation definitions (Reitz, Vries et al. 2009). The editor also is to allow for exporting created data models, for instance, in the form of XMI and INTERLIS-specific documents and as GML application schemas. The models can then be imported into HALE and be used there for creating transformation definitions.

The CST is intended to allow for transforming geospatial data based on conceptual schemas including database schemas, UML models, INTERLIS models, XML-based schemas and models defined using ontology languages such as OWL. The transformation definitions have to be provided to the CST as OML documents, e. g. by creating them beforehand using HALE (Fitzner et al. 2009). However, the actual implementation available of the CST only allows for transforming between source and target geospatial data provided as GML documents (Data Harmonisation Panel 2012), thus, requiring the transformation definitions to be created at the data format schema level between GML application schemas.

4.3.2 Transformation tools

Several commercial and open-source software tools for semantically transforming geospatial data exist. A still up-to-date overview and evaluation of the tools most prevalent in the geospatial domain can be found in (Beare, Howard et al. 2010). In the following, three selected tools will be discussed shortly, FME since it plays an important role in the remainder of this thesis and, furthermore, Go Publisher and HALE, to explain the transformation approaches implemented in the EU projects ORCHESTRA and HUMBOLDT, respectively (cf. section 4.3.1, page 63). FME and Go Publisher are proprietary software solutions, whereas HALE is an open-source tool.

The software FME (Feature Manipulation Engine) Desktop from the Canadian company Safe Software (Safe Software 2015a) is a Spatial ETL tool able to convert, transform and integrate geospatial as well as non-geospatial data in more than 300 formats. The transformation definitions, referred to as FME workspaces, are created by means of a graphical user interface (GUI), which is referred to as FME workbench. FME offers a huge number of spatial and non-spatial transformation functionalities which are made available within the GUI in the form of more than 400 so-called transformers; furthermore, additional transformation functionalities can be added by including user-defined Python and JavaScript scripts. Above that, FME provides APIs for the programming languages Java, C++ and Python which allow for implementing readers and writers for data formats and modelling languages not yet supported by FME and also for creating additional transformers. By making use of the APIs, it is also possible to specify complete transformation definitions beforehand using a platform-independent transformation language and to process them afterwards using FME, which was done in the mdWFS project (cf. section 4.3.3, page 67). FME works at the data format level, at the data format schema level and at the conceptual schema level. At the data format level, geospatial data are read and written. At the data format schema level, most of the schemas describing the source and target data are provided to FME and also the corresponding transformation definitions are defined at this level. In cases where no schema is available for the source data, the schema is deduced from the source data itself. At the conceptual schema level, conceptual data models can be provided to FME, such as INTERLIS models in the form of .ili documents (Eisenhut Informatik AG 2006) or UML models in the form of XMI documents, as demonstrated in the mdWFS project, the corresponding transformation definitions being then defined at this level. FME is based on the Relational paradigm; this means that internally FME maps all data models to the Relational paradigm and also represents them in this way in the GUI. With the software product FME Server it is even possible to transform geospatial data in a web-based environment. This functionality was employed in the mdWFS project to realise the transformations in a web-based environment.

The software tool Go Publisher from the British company Snowflake Software focuses on publishing geospatial data, stored in databases, in the form of XML-based data formats such as GML, KML or AIXM (Snowflake Software 2014). By means of Go Publisher, transformation definitions are always

defined between a database schema on the source side and an XML-based schema on the target side and, thus, at the data format schema level. The transformation definitions are defined within a GUI; internally, Go Publisher uses a proprietary transformation language (Beare, Howard et al. 2010). The transformation definitions can be complemented by custom mapping definitions written in SQL, XSLT or Java, which are executed after the actual transformation has been carried out. It is not possible to specify complete mapping definitions beforehand using a platform-independent transformation language and to process them afterwards using Go Publisher. Furthermore, Go Publisher provides an OGC Web Feature Service (WFS) implementation which can be used for realising a web service capable of executing transformations. This WFS implementation was, for instance, employed in the ORCHESTRA project for realising the FAS-X service.

The HUMBOLDT Alignment Editor (HALE) is a software tool which was originally developed as part of the HUMBOLDT project for specifying transformation definitions and transforming geospatial data. The transformation definitions are created within a GUI. Internally, HALE uses a transformation language which is based on OML/EDOAL (Reitz and Templer 2012) (cf. section 3.4.3, page 50). According to the HALE user guide, HALE differentiates between conceptual schemas, such as UML models and ontologies, and logical schemas, such as XML Schemas and database schemas (Data Harmonisation Panel 2015a). The user guide, furthermore, states that '[i]n *HALE* we define the schema mapping on the conceptual level' (Data Harmonisation Panel 2015a); however, only 'experimental support for UML (in XMI or Enterprise Architect encoding) and OWL' (Reitz and Templer 2012) seems to be provided up to now. Based on an examination of the currently downloadable HALE version 2.9.3 (Data Harmonisation Panel 2015b), the transformation definitions can only be created at the data format (schema) level between source and target models which are XML Schema documents including GML/CityGML application schemas, SQLite/SpatiaLite and PostgreSQL/PostGIS database schemas, Shapefile documents (only source), Excel spreadsheets, CSV documents (only source) or HALE Schema Definition documents. Transformation definitions can be exported in the form of XSLT documents, Excel spreadsheets, CSV documents and HALE Alignment documents, whereas an import of transformation definitions is possible in the form of HALE Alignment documents and OML/gOML documents based on the RDF-XML syntax. In this way, HALE allows for creating transformation definitions within HALE, but executing them using external transformation tools and vice versa.

4.3.3 The mdWFS approach of model-driven transformation of geospatial data

As can be seen from the explanations provided above on several projects and transformation tools dealing with the transformation of geospatial data, transformation definitions are still mostly specified at the data format (schema) level. This fact is underpinned by a workshop about schema transformation methods and tools conducted by INSPIRE KEN and EuroSDR in 2012 which covered the full range of today's prevailing transformation approaches in research and industry in numerous presentations (EuroGeographics 2012), from which the same conclusion can be drawn.

In some cases, such as in the project GiMoDig (cf. section 4.3.1, page 63), transformation definitions are indeed first defined at the conceptual schema level, actual implementations, however, require the transformation definitions to be specified again at the data format (schema) level, not being clear, whether the latter transformation definitions were created based on the former conceptual definitions.

The Technical Guidance on transformation services for INSPIRE (cf. section 4.3.1, page 63) even argues that '[a]lthough UML can be used to express a range of modelling abstractions (including

physical models) it cannot be used to express the actual data content. This means that if UML where used as the schema description language, the service would have to perform an internal translation of the UML into an appropriate concrete physical model of the datasets (like XSD). Although this process is tractable, [...] this extra level of conversion places unnecessary processing overheads on the service and introduces significant extra development complexity with no substantial benefit' (Howard et al. 2010). However, this argumentation contains one decisive weakness: The proposals given in the Technical Guidance regarding the use of GML and RIF restrict the applicability of transformation of geospatial data to a very specific and narrow use case, i. e. 'a network service that loads source data from a Web Feature Service (WFS) [...] or an FTP Site, performs a transformation based on a mapping definition and outputs the INSPIRE-schema compliant data to a Transactional WFS (WFS-T) [...] or an FTP Site' (Howard et al. 2010). The transformation definitions are not platform-independent, they rather are specific to the platform GML only. When intending to execute a transformation between other data formats – which might be required within the context of other use cases –, a reuse of the transformation definitions is not possible, as has already been discussed in section 3.3.2, page 39, in the context of the format-driven transformation.

A different approach was taken by the project *Model-driven approach for accessing distributed spatial data using Web Services – demonstrated for cross-border GIS applications (mdWFS)*, which was conducted by the Technische Universität München (TUM) and the Swiss Federal Institute of Technology Zurich (ETH Zurich) on behalf of the German Federal Agency for Cartography and Geodesy (BKG) and the Swiss Federal Office of Topography (swisstopo) from 2006 until the end of 2011 (Staub 2007; Donaubauer, Staub et al. 2008; Staub et al. 2008; Staub 2009; Donaubauer, Kutzner et al. 2010; Fichtinger 2011). The approach chosen in the mdWFS project fully complies to the concept of model-driven transformation of geospatial data as introduced in section 3.3.2, page 39, and in figure 3.10, page 41, which means that the transformation definitions are defined at the conceptual schema level between conceptual source and target models created using a formal modelling language. The models used within the project were defined using the modelling languages INTERLIS and UML. To be able to specify the transformation definitions between these models, the transformation language UMLT was developed within the project (cf. section 3.4.2, page 46, for a detailed description of UMLT). Furthermore, to allow for executing transformations in a web-based environment, a model-driven Web Feature Service (mdWFS) was defined as an extension of the OGC WFS specification. A DoTransform operation serves in transforming UML models and UMLT definitions sent to the service in the form of XMI documents. Afterwards, the transformed data are accessible via a newly configured WFS instance.

In the course of the research project, two prototypes were developed. Whereas the first implementation solely aimed at demonstrating the feasibility of the approach, the second implementation took into account the necessity for data providers to provide data conforming to the INSPIRE data specifications. In addition, platform-specific requirements arising from the project sponsors' production environment were considered and the experience gained from the first prototype was incorporated. The author of this thesis was involved in the development of the second prototype.

4.3.3.1 First mdWFS prototype

The first mdWFS prototype was developed based on an Oracle Spatial database (Oracle 2015), a WFS implementation of the deegree project (deegree.org 2015) and INTERLIS-specific tools (*INTERLIS - The GeoLanguage* 2015). The transformation language UMLT was implemented using the Eclipse Modeling Framework (EMF) (Steinberg et al. 2009) and UML 2, an EMF-based implementation of

the modelling language UML 2 for the Eclipse platform (The Eclipse Foundation 2015e). UML 2 was extended by the UMLT language constructs presented in section 3.4.2, page 46. The transformation of the geospatial data was executed by means of SQL statements and Java instructions. The prototype was tested by transforming the German Digital Landscape Model (ATKIS Base-DLM) as part of the AAA reference model and the digital municipal boundaries of Switzerland (GG25) to the draft EuroSpec Feature Catalogue Administrative Boundaries. The transformation definitions between the models were specified using the UMLT HUTN syntax. However, since the UMLT HUTN syntax is based on INTERLIS, the German and the EuroSpec models had to be remodelled using INTERLIS to be able to execute the transformations. An INTERLIS parser was used to create XMI documents from the INTERLIS models, and a UMLT HUTN parser was specifically developed for automatically converting the UMLT HUTN documents into XMI documents.

4.3.3.2 Second mdWFS prototype

Since the remodelling of UML models in the form of INTERLIS models involves an unrealistic effort when applied in a production environment, a solution was needed which is independent from the tight coupling of the UMLT HUTN syntax and of the UMLT HUTN parser with the modelling language INTERLIS. The decision was made to develop a graphical *UMLT editor*, since such an editor solely requires knowledge of the UMLT concepts and, in addition, allows for specifying the transformation definitions in a user-friendly way. The graphical UMLT editor was implemented based on EMF and the Eclipse Graphical Modeling Framework (GMF) (The Eclipse Foundation 2015d), making use of the UMLT metamodel developments from the first prototype (Elfouly and Kutzner 2013). By means of this editor, the transformation rules can now be defined visually, and can afterwards be exported as XMI documents for further use as platform-independent input to any transformation tool able to interpret XMI.

Furthermore, a solution equally powerful than SQL statements and Java instructions was sought for the implementation of the transformation functionalities. Since the development of a completely new transformation module is very complex and time-consuming, the integration of an existing software solution into the second prototype was regarded as more advantageous. The software FME (cf. section 4.3.2, page 66) proved to be a suitable platform for providing the transformation functionalities within the mdWFS. For this reason, FME was extended in such a way that it can handle the UML models and the UMLT transformation definitions in the form of XMI documents.

To be able to read the UML models, an *XMI Reader* plug-in was developed for FME which makes use of the FME Java API, of EMF and in particular of UML 2. By means of this XMI Reader, FME is able to parse the source and destination models and to convert them into the internal FME structure. Since FME is based on the Relational paradigm and UML on the OO paradigm, some difficulties in mapping UML models to the FME structure had to be overcome, especially regarding the OO-specific constructs association and inheritance, which are not available in the Relational paradigm. The following rules, for instance, were defined for mapping the concepts feature type, association, complex type and inheritance to the FME structure: All UML classes representing feature types (indicated by the stereotype «featureType») are mapped one-to-one to corresponding FME feature types (represented in the FME workbench by corresponding FME feature type boxes), except abstract UML feature types, they are not mapped. For abstract UML feature types as well as for inheritance in general, a subclass strategy is applied, which means, the attributes from the UML superclass are added to the concrete FME feature types inheriting from the corresponding UML superclass. Associations between feature types, including reflexive associations, are mapped to additional feature type boxes

in FME and are given the name A__FeatureType1FeatureType2, FeatureType1 and FeatureType2 being the names of the feature types associated with each other. In addition, attributes describing the associations in more detail (role names, multiplicities and directionality) are assigned to the association boxes. Attributes with complex types are resolved by representing them in FME using point notation, which, however, can result in numerous attributes, a feature type can possess.

Above that, a second plug-in was developed for FME which implements a transformer called *UMLTApplier*. This transformer parses the UMLT transformation definitions and maps them onto the internal transformation functionalities of FME. For each UMLT function, an FME factory pipeline (Safe Software 2005) is created which contains exactly those FME factories and functions (Safe Software 2015b) which reflect the functionality of the corresponding UMLT function. These factory pipelines transform the geospatial data whenever the actual transformation is invoked. The plug-in makes again use of the FME Java API, of EMF and of UML 2. Please refer to appendix D.4 for an overview of the UMLT functions implemented within FME up to now. When opened in an FME workbench, the whole FME workspace required for transforming geospatial data in a model-driven way simply consists of two XMI Readers, one reading the source model and the other one reading the target model, and the UMLTApplier transformer.

To be able to transform the geospatial data by means of the mdWFS service interface, a deegree-based OGC Web Processing Service (WPS) implementation jointly developed by Safe Software and the German company lat/lon was employed[5]. This WPS implementation uses FME as process provider, which means that FME workspaces available via FME Server can simply be executed by requesting the Execute operation of the WPS interface; thus, allowing for executing the mdWFS FME workspace in a web-based environment. The operations of the mdWFS interface were emulated by implementing them in the form of WPS processes, which means, when requesting the WPS Execute operation, the name of the mdWFS operation to be executed needs to be stated as process parameter. Furthermore, a simple web client was developed which allows for communicating with the mdWFS service interface in a user-friendly way.

The second mdWFS prototype was tested by transforming the German Digital Landscape Model (ATKIS Base-DLM) as part of the AAA reference model and the Swiss Topographic Landscape Model (TLM) to the harmonised INSPIRE data specifications, focusing on the INSPIRE themes Protected Sites and Hydrography. Both, the source models as well as the target model, are available as UML models.

4.3.4 Further academic research works

In addition to the transformation projects and tools presented above, in academia several other research works exist which deal with modelling and model-driven transformation of geospatial data. In this section, an overview of relevant works is provided; this section, furthermore, takes into account two other topics the remainder of this thesis will focus on, the transformation between UML profiles and the use of UML package merge for creating UML profiles.

(Balley et al. 2006) presents a system which enables users to interactively restructure existing geospatial data in such a way that the structure fits the users' needs. The approach requires the geospatial data to be represented by three different types of schemas, a conceptual schema, a logical schema and a physical schema. The terminology is strongly influenced by the DBMS technical space; in fact, these schemas are equivalent to the PIM, PSM and PM models in the MDA technical space.

[5]This WPS implementation was never publicly released, but lat/lon and Safe Software willingly agreed on providing the implementation for the purposes of the research project mdWFS.

The user defines the individual transformation steps based on a view of the conceptual schema which is displayed 'in a UML-like form' (Balley et al. 2006) within a GUI. When the actual transformation is invoked, the individual transformation steps are applied to all three schemas and the restructured geospatial data is delivered to the user. This approach differs from all other transformation approaches presented up to now in the fact that the transformation is based solely on the source schema, whereas the target schema exists only implicitly, i. e. in the mind of the user. The user needs to know exactly what the resulting structure has to look like and has to pay attention throughout the definition of the individual transformation steps that in the end he obtains the desired structure. The user cannot simply provide a target schema and create the transformation definitions between the source schema provided by the system and the target schema provided by the user.

The work of (Staub 2009) proves the feasibility of the concept of model-driven information integration of geospatial data, as presented in section 3.3.2, page 39, based on the project mdWFS which was presented in the previous section. Furthermore, the work identifies several limitations of semantic interoperability; these include limited meta-interoperability due to source and target models complying to different metamodels – a problem which is tackled in the remainder of this thesis –, non-bijectivity of semantic transformations, the existence of too many different encoding rules, the use of different semantics for describing the same UoD in different data models as well as source and target models defining non-overlapping information. The work, furthermore, discusses existing approaches from the field of ontologies and classifies the model-driven approach into the ontology spectrum of Obrst (cf. section 2.3.4, page 17).

(Fichtinger 2011) developed feature-based classifications of transformation approaches and of the types of semantic heterogeneity which can occur between source and target models and, furthermore, derived from these heterogeneities functions required for the creation of suitable transformation definitions. The work, in addition, discussed the problem of non-overlapping information by analysing whether the information provided by the source schemas used in the work is sufficient for covering the INSPIRE theme Hydrology completely. Above that, the work pointed out differences, possibilities and limitations of semantic transformation at the conceptual schema level and at the data format schema level. The classifications and analyses were conducted based on the research projects HUMBOLDT (cf. section 4.3.1, page 63) and mdWFS (cf. section 4.3.3, page 67).

The work of (Schulze Althoff 2011) evaluates the applicability of the MDA approach in the geospatial domain. The work analyses whether existing geospatial modelling languages as well as new modelling languages, in particular DSLs for the geospatial domain, can be (re-)defined based on a common meta-metamodel. Furthermore, it is analysed whether standard transformation tools can be employed for executing horizontal transformations between data models defined using different metamodels, and also whether vertical transformations can be executed on these data models. The analyses were based on Ecore as the common meta-metamodel and were verified practically by redefining the modelling languages INTERLIS 2 and MADS (Modeling of Application Data with Spatio-temporal Features) as Ecore-based metamodels. Furthermore, the HUMBOLDT Modelling Language (HML), a DSL, was developed (cf. section 4.3.1, page 63), and a horizontal transformation was conducted between HML and UML as well as a vertical transformation between HML and GML Schema. It is argued that the advantage in using DSLs lies in the fact that they 'can be efficiently adjusted to the intended user communities, [...] can be clean from superfluous elements and avoid complex constructs [..., and] guide to clean, compact and precise models' (Schulze Althoff 2011). Thus, DSLs allow for generating specialised tools such as editors for defining data models which only provide those elements defined within the DSL. This was verified by generating the HUMBOLDT GeoModel editor automatically from the HML metamodel. Furthermore, it is argued that by basing all

modelling languages on the same meta-metamodel, data models can first be defined using DSLs and can then be transformed into data models based on metamodels for which tools for further processing exist. The work does not take into account UML profiles, however, since UML profiles turn UML into a DSL (cf. section 2.2.4, page 11), the work exhibits some similarities with the approach proposed in this thesis.

As part of a work dealing with information discovery in SDIs and the model-driven generation of feature catalogues, (Einspanier 2005) developed a DSL which represents the GFM as a well-defined, formal MOF-based metamodel with a precise and mathematically defined semantics. Furthermore, the work formally defines rules for mapping UML application schemas, i. e. models defined using the UML metamodel, into GFM application schemas, i. e. models defined using the MOF-based GFM metamodel. The development of this DSL was seen as necessary since the definition of the GFM, as provided by the standard ISO 19109, does not conform to strict metamodelling and, thus, poses problems which prevent the GFM from being deployable in model-driven development. In terms of the four-layer metamodel architecture of UML, the GFM represents a metamodel at the layer M2, however, it was not defined using a meta-metamodel at the layer M3, but by means of the UML metamodel which itself is located at the layer M2. Furthermore, constraints related to the abstract syntax of the GFM are only defined informally, also the semantics is informal and incomplete. The GFM does not define a concrete syntax, thus, the concepts of the GFM have to be mapped to a modelling language providing a concrete syntax, which is why the standard ISO 19109 provides mapping rules for mapping the GFM to UML, however, these rules are only defined informally (cf. section 2.3.2, page 14, as well). The work does not deal with UML profiles, however, it is stated that '[f]or better enabling language integration and reuse of models [...], a clear and unambiguous profile would be required. This could e.g. be achieved by providing more specific stereotypes that properly identify the corresponding GFM constructs' (Einspanier 2005). In this way, UML would be able to represent the semantics of the GFM more precisely and in more detail.

With GeoProfile, a distinct formally defined UML profile for specifying geographic databases at the conceptual schema level exists (Lisboa-Filho, Sampaio et al. 2010). The concepts of the GeoProfile are based on several conceptual models published in scientific literature for designing geographical databases, in particular OMT-G (Object Modeling Technique for Geographic Applications), MADS, GeoOOA (Object-oriented Analysis for Geographic Information Systems), UML-GeoFrame and the Perceptory's model. GeoProfile does not make use of stereotypes defined in the standards ISO/TS 19103 and ISO 19136 (cf. sections 5.1, page 76, and 5.3, page 86, respectively), it rather represents a DSL of its own. In (Lisboa-Filho, Nalon et al. 2013) the GeoProfile was applied in designing a database based on the MDA approach, by defining a Computation Independent Model (CIM) model using GeoProfile. The CIM model was then transformed into a PIM model which takes into account concepts of some standards from the ISO 191xx series such as the geometries from the standard ISO 19107. The concepts of the standards ISO/TS 19103 and ISO 19109, however, are not considered. Afterwards, the PIM model was transformed into a PSM model targeted at an object-relational platform. Both transformations were conducted using ATL.

Up to now, no scientific publications exist in the field of geographic information science which explicitly take into account UML profiles in the context of model transformation or information integration of geospatial data. In computer science literature, in contrast, a few contributions dealing with the transformation between UML profiles can be found.

Based on the fact that different aspects of a complex embedded software system often are described by means of different models, each of them created using a different DSL (or UML profile), difficulties can arise when these models shall be used jointly. The approach presented in (Noyrit et al. 2010) tries

to overcome this obstacle by proposing the definition of a new UML profile which includes exactly those elements from the original UML profiles required for the intended combined use and which can be complemented by additionally required elements not present in the original UML profiles. Starting with a domain model (cf. section 6.3, page 123) which defines all those concepts required in the new UML profile, an interactive process follows in which the original UML profiles are aligned with the new UML profile by means of aligning the concepts, the abstract syntax and the concrete syntax. The approach is demonstrated by combining the two standard UML profiles MARTE (a UML profile for designing embedded systems) and SysML (a UML profile for Systems Engineering) in this way. Furthermore, the approach also allows for transforming models defined using the new UML profile into models conforming to the original profiles.

Within the same domain, a work from (Riccobene and Scandurra 2012) deals with a vertical transformation between the SysML UML profile, which is located at the PIM level, and the SystemC UML profile (a UML profile for the programming language SystemC), which is located at the PSM level, by directly defining mappings between the two UML profiles. These mappings are then used for transforming PIM models, to which the SysML UML profile is applied, into PSM models, to which the SystemC UML profile is applied.

The work of (Eessaar 2008) presents an approach for creating new UML profiles from already existing UML profiles, motivating the approach with the argument of saving time in this way, compared to having to create a UML profile from the beginning. The profile created by means of this approach, however, represents just a first draft which then needs to be refined and complemented manually; this is, for instance, necessary when the target UML profile requires concepts which do not exist in the source UML profile. In a first step, mappings are defined between elements from the source metamodel and elements from the target metamodel, the metamodels representing domain models which are modelled using UML. Afterwards, the target UML profile is created by transforming the source UML profile based on the metamodels and the mappings defined. The approach is tested by transforming a UML profile representing the object-relational model underlying the SQL:2003 standard into a UML profile representing the object-relational model proposed by The Third Manifesto.

Scientific publications addressing the generation of UML profiles by using UML package merge (cf. section 2.4.3, page 23) do not seem to exist up to now.

5 Critical examination and proposed improvement of UML profiles commonly used in the geospatial domain

One of the limitations of semantic interoperability identified by (Staub 2009) (cf. section 4.3.4, page 70) is the limited meta-interoperability due to source and target models complying to different metamodels. Similarly, section 4.2.1.2, page 57, states that UML profiles which involve a semantic modification of the UML specification rather represent a new modelling language and, thus, can cause problems when transforming UML models to which these UML profiles have been applied. Above that, also community-specific UML profiles can hinder transformation in a cross-community environment when they exhibit semantics not present in other communities. Therefore, this chapter takes a closer look at currently existing UML profiles, in particular the UML profiles defined by the standards ISO/TS 19103, ISO 19109 and ISO 19136. Many SDI initiatives require the specification of UML data models compliant to these ISO standards and, thus, to the UML profiles defined therein. Moreover, these SDI initiatives often extend the ISO UML profiles by additional concepts specific to the individual SDI. For this reason, also the UML profiles of the INSPIRE initiative, of the German AAA reference model, of the European Location Framework (ELF) and of the Swiss modelling language INTERLIS will be considered in this examination. In addition, also the UML profile defined by the CityGML community will be examined.

The examination of ISO/TS 19103 UML profile, the ISO 19136 UML profile and the INSPIRE UML profile is split into two parts. The first part discusses the stereotypes as they are currently defined in the UML profile. The discussion takes into account that the stereotypes are defined against the background of UML 1 where it is allowed to use stereotypes and tag definitions without a formally defined UML profile, but at the same time also points out the problems which occur when they are defined in the context of UML 2. Given the fact that the revisions of the standards ISO/TS 19103 and ISO 19109 will be based on UML 2 and that also INSPIRE already uses UML 2, the second part proposes a formally defined UML profile compliant to the UML 2 profile definition of the OMG which eliminates the shortcomings exposed in the first part, illustrating in this way, how the stereotypes can be defined compatible with UML 2 without loosing the semantics the stereotypes exhibit. Each UML profile discussed and proposed is accompanied by a UML profile diagram. UML profile diagrams provide a good overview of the available stereotypes and tag definitions, support the modeller in defining UML application schemas, and should, therefore, always be provided. The other UML profiles are discussed more condensed, since they reuse many concepts from the ISO/TS 19103 UML profile, the ISO 19136 UML profile and the INSPIRE UML profile and, thus, exhibit some similarities.

Within the following discussion, the spelling of the stereotypes is adopted as is defined in the corresponding ISO standards or SDI initiatives. For the proposed formal UML profiles, however, upper case spelling is consistently used as is suggested in the UML specification (cf. section 2.4.1, page 19).

5.1 The UML profile of the standard ISO/TS 19103

An essential standard in the ISO 191xx series of geographic information standards is the standard *ISO/TS 19103:2005 Geographic information — Conceptual schema language* (ISO 2005c). The standard defines UML together with OCL and a set of basic data types as the modelling language to be used for modelling geographic information and specifies rules and guidelines for how to use UML appropriately. The rules and guidelines are based on UML version 1.4.2, which is defined in the standard ISO/IEC 19501.

The standard ISO/TS 19103 specifies the general use of UML elements, such as classes, attributes, operations, relationships and associations, and also provides information regarding naming conventions, cardinalities and role names, the optionality of attributes and role names, and the use of UML packages. Above that, the standard defines a set of elementary data types, such as Date and Time, Numerics (e. g. Integer, Real), Text (e. g. CharacterString), Truth (e. g. Boolean) and Units of Measure (e. g. Length, Scale), since UML version 1.4.2 does neither define nor prescribe any specific data types. UML 2, in contrast, introduces four primitive data types (Integer, String, UnlimitedNatural and Boolean).

Of importance for the discussion in this chapter is that the standard ISO/TS 19103 defines three UML stereotypes: «CodeList», «Union» and «Leaf». The textual definitions of the stereotypes are listed in table 5.1. Formal definitions of the stereotypes as part of a UML profile are not provided by the standard ISO/TS 19103. Rather, the standard refers to the complete set of rules and guidelines on the use of UML for modelling geographic information as a UML profile – which goes far beyond the scope of what the UML profile definition of the OMG defines as a UML profile. In addition, the standard lists and uses several stereotypes which are predefined in UML 1.4.2. These stereotypes are: «Interface», «Type», «Control», «Entity», «Boundary», «Enumeration», «Exception», «MetaClass» and «DataType»[1].

Stereotypes are regarded by the standard ISO/TC 19103 not only as a means to adapt UML to the domain of geographic information but also ‘as flags to language compilers to determine how to create implementation models from the abstract [model]’ (ISO 2005c). Related to the encoding of geospatial data (cf. section 3.3.1, page 36), this means that the stereotypes tell the transformation tool in which way model elements from application schemas are to be represented in implementation schemas or data transfer format schemas.

The following sections will discuss some of the stereotypes in more detail, point out the deficits of the UML profile defined by the standard ISO/TC 19103 and finally propose a formal definition of the stereotypes as part of a UML 2 profile compliant with the UML profile definition of the OMG.

5.1.1 Discussion of the ISO/TS 19103 stereotypes

First, the ISO/TS 19103 definitions of the stereotypes «CodeList» and «Union», quoted in table 5.1, will be analysed in more detail. The definitions are provided in a textual and informal way which leads to an ambiguous understanding regarding which UML metaclasses the stereotypes extend. Furthermore, the discussion will show that the semantics of the stereotypes is not consistent with

[1] The stereotypes «Control» and «Boundary» are actually defined in UML 1.4.2 as part of a UML example profile for Software Development Processes, whereas the stereotype «Exception» cannot be found at all in the UML 1.4.2 specification. Also, these three stereotypes are not included in UML 2 any more, whereas the other stereotypes from this list are still provided. Furthermore, it is important to note that «Interface», «Enumeration» and «DataType» are actually UML keywords and not stereotypes (cf. section 2.4.2, page 22).

Table 5.1: Stereotypes defined by the standard ISO/TS 19103

Stereotype	Definition
«CodeList»	Defines 'a flexible enumeration that uses string values through a binding of the Dictionary type key and return values as string types; e.g. Dictionary (String, String). If the elements of a list are completely known, an enumeration shall be used; if only the likely values of the elements are known, a code list shall be used' (ISO 2005c). This means that by using a code list, the value set can be extended arbitrarily by the user, whereas a UML enumeration defines a fixed value set within the UML model. Furthermore, the values of a code list are defined as key-value pairs, whereas an enumeration contains only values, which are referred to as enumeration literals (ISO 2012b).
«Union»	Defines a type which consists 'of one and only one of several alternatives (listed as member attributes)' (ISO 2005c). Thus, a type marked as union is allowed to have at run-time exactly one of the defined attributes as value only.
«Leaf»	Denotes a UML package which does not contain any other UML packages, but solely definitions in the form of UML class diagrams (ISO 2005c).

the semantics of the UML metaclasses they extend, which means, the stereotype definitions modify the UML metamodel in such a way that a new modelling language is created. Afterwards, it will be discussed why «Enumeration» and «DataType» are incorrectly referred to as stereotypes by the standard ISO/TC 19103. The discussion of the stereotype «Leaf» will take place in section 5.1.2, page 81, since its definition is non-ambiguous and its semantics is consistent with the semantics of the UML metaclass it extends.

5.1.1.1 The stereotype «CodeList»

Defining a stereotype formally within a UML profile makes apparent at first view, which UML metaclass the stereotype is an extension of. However, the textual definition given for the stereotype «CodeList» in table 5.1 does not indicate which UML metaclass this stereotype extends. For this reason, we take a closer look at the following descriptions in the standard ISO/TS 19103:

1. A code list is a 'flexible *enumeration*' whose key-value pairs are to be modelled 'as *attributes* with a stereotyped «CodeList» with an *attribute name* for each value and the code(key) represented as an *initial value*. In the case when only attribute names are shown, the codes and their descriptions are equivalent [emphases added]' (ISO 2005c).
2. An enumeration 'is a user-definable data type whose instances form a list of named literal values'; furthermore, enumerations 'are modelled as *classes* that are stereotyped as «Enumeration» [emphasis added]' (ISO 2005c).

From the first description alone it is difficult to conclude which UML metamodel element to use for modelling code lists. However, taking into account the second description, one can conclude that code lists are to be modelled as UML classes, since according to (1) code lists are enumerations and according to (2) enumerations are modelled as classes; therefore, code lists are to be modelled as classes as well. Furthermore, the first description speaks of attributes and initial values, both appear in UML classes. According to this argumentation the stereotype «CodeList» extends the UML metaclass *Class*.

Applying a stereotype to a UML model element does not change the semantics of the UML model element. This means, a UML class to which the stereotype «CodeList» has been applied, still behaves like a UML class. This UML class still contains attributes with types, even if the types are now, since the UML class is used as a code list, not provided. This fact is illustrated by using an example from the INSPIRE data specification Administrative Units. In this data specification the code list *AdministrativeHierarchyLevel*, depicted on the left side of figure 5.1, is defined. In accordance with

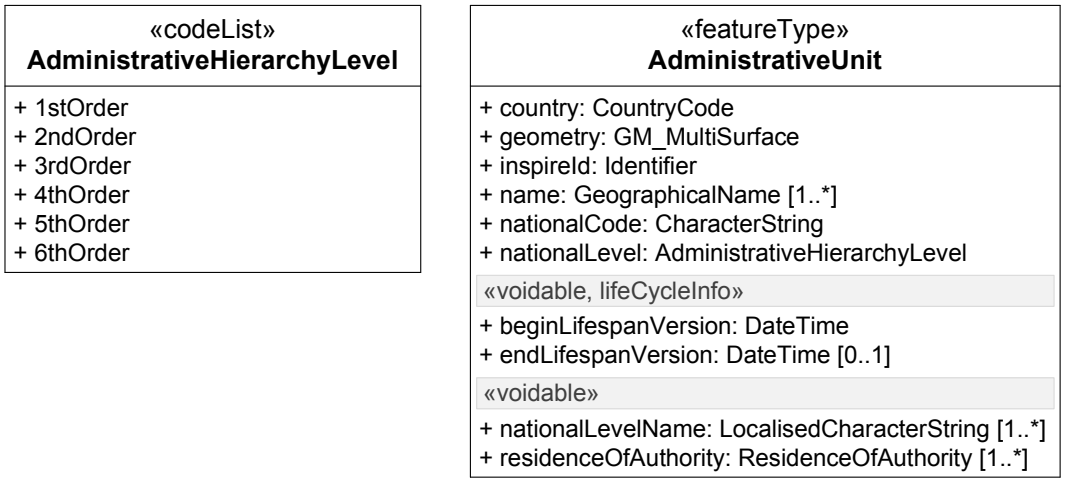


Figure 5.1: Code list *AdministrativeHierarchyLevel* and feature type *AdministrativeUnit* from the INSPIRE data specification Administrative Units (JRC 2014d)

the above argumentation the code list is represented as a UML class. Therefore, in the semantics of the UML specification the values *1stOrder*, *2ndOrder*, ..., *6thOrder* represent attributes of this UML class. This can also clearly be seen from the XMI representation of the code list in listing 5.1, where the type *uml:Property* indicates this fact. Furthermore, the UML specification states that attributes for which no multiplicity is defined are by default of multiplicity 1. This also holds for the attributes *1stOrder*, *2ndOrder*, etc. In addition, since no types and no initial values are provided for the attributes, they 'may represent values of any type' (ISO 2012b) according to the UML specification. A UML tool is not able to recognise that the values of the attributes shall be identical to their attribute names. Thus, at the data format level, an instance of *AdministrativeHierarchyLevel* would mandatorily possess the attributes *1stOrder*, *2ndOrder*, ... *6thOrder* and the value of each attribute could be an arbitrary value of any type.

Listing 5.1: XMI representation of the code list *AdministrativeHierarchyLevel*

```
<packagedElement xmi:type="uml:Class" name="AdministrativeHierarchyLevel">
  <ownedAttribute xmi:type="uml:Property" name="1stOrder"/>
  <ownedAttribute xmi:type="uml:Property" name="2ndOrder"/>
  <ownedAttribute xmi:type="uml:Property" name="3rdOrder"/>
  <ownedAttribute xmi:type="uml:Property" name="4thOrder"/>
  <ownedAttribute xmi:type="uml:Property" name="5thOrder"/>
  <ownedAttribute xmi:type="uml:Property" name="6thOrder"/>
</packagedElement>
```

The representation of code lists as UML classes also affects their use as attribute types. Consider the feature type *AdministrativeUnit* on the right side of figure 5.1. It contains the attribute *nationalLevel*

which is of the type *AdministrativeHierarchyLevel*. The idea behind code lists is that at the data format level one of the code list keys will be assigned to the attribute *nationalLevel* as attribute value. However, due to the representation of the code list as a UML class, the attribute value of *nationalLevel* is not a code list value, but an instance of the class *AdministrativeHierarchyLevel* itself in the form of a complex type.

When using the profile mechanism as defined by the UML specification, it is not possible to change the semantics of the UML metamodel (cf. section 2.4.1, page 19). However, forcing a UML class to behave compliant to the semantics of the stereotype «CodeList» as defined by the standard ISO/TC 19103, results in such a semantic modification of the UML metamodel and the stereotype does not simply represent any more an extension of the UML metaclass *Class*.

5.1.1.2 The stereotype «Union»

According to the standard ISO/TC 19103, the stereotype «Union» denotes a type which can define several attributes, but which at the data format level is allowed to exhibit one and only one of the attributes defined. From a UML specification point of view, however, an instance of a type cannot just exhibit one attribute, when it was defined differently for the type itself. This fact is illustrated by using an example from the AAA reference model. Figure 5.2 shows on the left side the union *AA_UUID* which has two attributes, *UUID* and *UUIDundZeit*. Both attributes are modelled with multiplicity 1. Thus, at the data format level both attributes have to appear and have to be assigned values to.

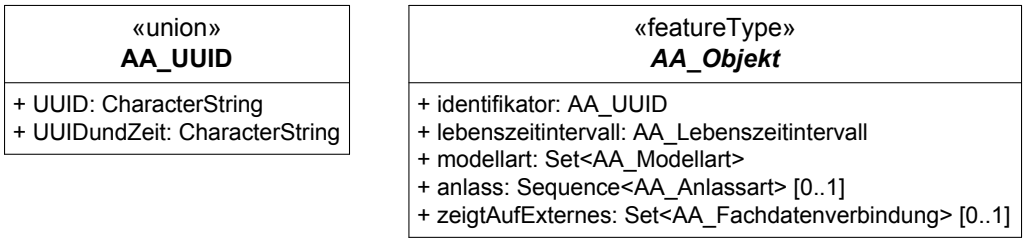


Figure 5.2: Union *AA_UUID* and feature type *AA_Objekt* from the AAA reference model (AdV 2009)

This representation also affects the use of unions as attribute types. Consider the feature type *AA_Objekt* on the right side of figure 5.2. It contains the attribute *identifikator* which is of the type *AA_UUID*. The idea behind unions is that at the data format level the attribute *identifikator* would receive its value either from the attribute *UUID* or from the attribute *UUIDundZeit*. However, due to the representation of the attributes with multiplicity 1, both attributes have to appear in an instance of the union and have to be assigned values to.

Furthermore, the textual definition of the stereotype «Union» in the standard ISO/TC 19103 does not expose immediately which UML metaclass the stereotype is an extension of – it simply refers to a union as 'a type' (ISO 2005c). The following three possibilities should be taken into consideration:

1. Since the standard speaks of 'a type', one obvious possibility would be that the stereotype «Union» extends the stereotype «Type» from the UML StandardProfileL2. The stereotype «Type» is defined there as a 'class that specifies a domain of objects together with the operations applicable to the objects, without defining the physical implementation of those objects. However, it may have attributes and associations' (ISO 2012b). This definition does not fit quite well the intended use of a union, since the semantics of a union indeed comprises the physical implementation of its

instances. In contrast, a UML type defines objects exclusively at an abstract level and usually requires an *ImplementationClass* which defines the concrete implementation of the type and which is related to the type through a *Realization* dependency (Hitz et al. 2005).

2. The stereotype «Union» could also be an extension of the UML metaclass *DataType*. A data type is defined in the UML specification as 'a type whose instances are identified only by their value. A data type may contain attributes to support the modelling of structured data types' (ISO 2012b). This means that instances of a UML data type do not exhibit an identity, in contrast to instances of a UML class. Therefore, instances of a data type are also referred to as values, whereas instances of a class are referred to as objects (Hitz et al. 2005). This definition of a data type reflects much more the semantics of a union, since a union primarily provides a value at the data format level and does not require to exhibit an identity. Thus, *DataType* represents an eligible UML metaclass of which the stereotype «Union» could be defined as an extension.
3. A third option would be to define the stereotype «Union» as an extension of the UML metaclass *Class*. This approach is used in current implementations of the UML profiles *UML Profile for GML Applications Schemas* and *UML Profile for INSPIRE Data Specifications* for use with the software Enterprise Architect (cf. appendix C.1, page 179). The disadvantage of this approach is that instances of a union always have an identity which, however, does not seem likely with respect to the semantics of the stereotype «Union» provided above.

5.1.1.3 The stereotypes «Enumeration» and «DataType»

As mentioned above, the standard ISO/TC 19103 makes use of the stereotype «Enumeration» and defines an enumeration as a 'user-definable data type whose instances form a list of named literal values'. This definition corresponds to the definition given by UML 1.4.2 for UML enumerations. Furthermore, the standard ISO/TC 19103 states that enumerations 'are modelled as classes that are stereotyped as «Enumeration»' (ISO 2005c), which implies that the stereotype «Enumeration» is defined as an extension of the UML metaclass *Class*. This statement, however, contradicts the modelling of UML enumerations as specified by UML 1.4.2. The UML 1.4.2 specification provides within the UML metamodel a distinct metaclass *Enumeration* which is a subclass of the metaclass *DataType* (this also holds for UML 2, cf. figure A.3, page 173). Therefore, «Enumeration» does not represent a stereotype, but a keyword (cf. section 2.4.2, page 22). No information is given in the standard ISO/TC 19103, why in UML models for geographic information enumerations are to be modelled differently, i. e. based on the metaclass *Class* instead of the metaclass *Enumeration*.

The same applies to the stereotype «DataType» which, in fact, is a keyword as well, since the UML 1.4.2 specification provides a distinct metaclass *DataType* which is derived from the metaclass *Classifier* (cf. section 2.4.2, page 22) (this also holds for UML 2, cf. figure A.3, page 173).

5.1.1.4 Conclusion

The UML profile mechanism only allows for extending the UML metamodel – but not for modifying its semantics – by defining stereotypes which constrain the UML metamodel elements within the limits of the semantics specified by the UML metamodel (cf. section 2.4.1, page 19). However, the above discussion shows that imposing the semantics of the stereotype «CodeList» on the metaclass *Class* results in such a semantic modification. The stereotype does not any more constrain the UML metaclass *Class*, but rather extends the UML language by defining a new modelling element. Thus, the stereotype «CodeList» can only syntactically be regarded as a UML class, since it is applied to the

UML class symbol; semantically, however, it represents an independent UML metamodel element and rather corresponds to a UML keyword (cf. section 2.4.2, page 22).

The same applies to the stereotype «Union», irrespective of which metamodel element the stereotype extends. It always represents an independent UML metamodel element which exhibits its own semantics and uses the UML class symbol only syntactically.

Taken as a whole, the UML profile defined by the standard ISO/TC 19103 cannot be regarded as a real UML profile in terms of being compliant with the UML profile definition of the OMG, it rather represents a new modelling language which exhibits the syntax of UML, but not its semantics. This also includes the rules and guidelines as well as the elementary data types defined by the standard ISO/TS 19103 which, according to the UML specification, are not part of a UML profile.

5.1.2 Proposed formal definition of an ISO/TS 19103 UML profile

Based on the above discussion, a formal UML profile is proposed in this section which defines the stereotypes «CodeList», «Union» and «Leaf» from the standard ISO/TC 19103 compliant with the UML 2 profile definition of the OMG, i. e. without causing a semantic modification of the UML metamodel. Figure 5.3 shows the formal UML profile. Each stereotype will be presented in more detail below.

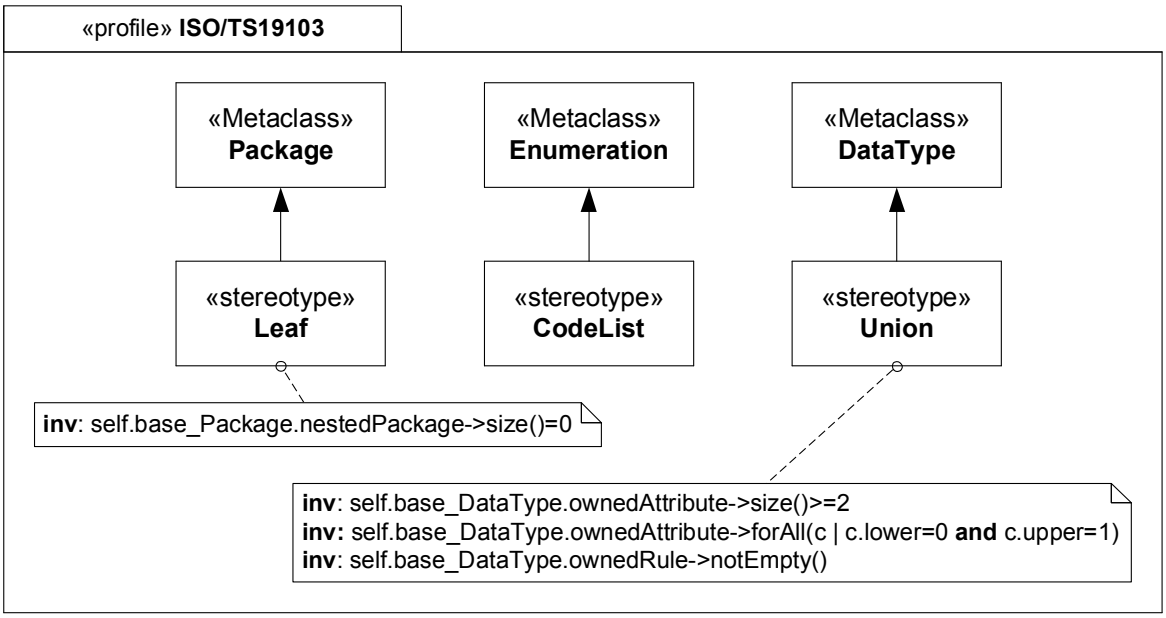


Figure 5.3: Proposed formal definition of an ISO/TS 19103 UML profile

As regards «Enumeration» and «DataType», section 5.1.1.3, page 80, explained why they, in fact, are UML keywords and not stereotypes. No reason is given in the standard ISO/TC 19103 why enumerations and data types should be modelled based on the UML metaclass *Class*. This approach would rather impose another semantic modification on the UML metamodel. Therefore, it is advised to use the corresponding UML metaclasses *Enumeration* and *DataType* when creating UML models.

5.1.2.1 Formal definition of the stereotype «CodeList»

Instances of an enumeration 'form a list of named literal values' (ISO 2005c). Since a code list denotes a 'flexible enumeration' (ISO 2005c), instances of a code list can equally be considered to represent a list of literal values. It is, therefore, proposed to define the stereotype «CodeList» as an extension of the UML metaclass *Enumeration*, as is depicted in figure 5.3. The key-value pairs of the code list can be represented as literal values by concatenating each pair of attribute name (value) and initial value (key) in the form of *value=key*, or when only the attribute name is given in the code list, by adopting this attribute name as literal value.

This approach is illustrated by using the code list *AdministrativeHierarchyLevel* from section 5.1.1.1, page 77, again. Figure 5.4 shows the new code list. It is now modelled as an enumeration to which the stereotype «CodeList» has been applied; thus, the code list exhibits now, on the one hand, the keyword «enumeration» and, on the other hand, the stereotype «CodeList». Furthermore, from the XMI representation of the new code list in listing 5.2 it can clearly be seen that the values *1stOrder*, *2ndOrder*, ..., *6thOrder* are now represented as a list of literal values, indicated by the type *uml:EnumerationLiteral*, and not as attributes any more.

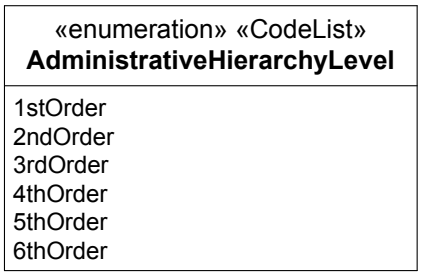


Figure 5.4: Code list *AdministrativeHierarchyLevel* from the INSPIRE data specification Administrative Units (JRC 2014d) based on the proposed formal definition of the stereotype «CodeList»

Listing 5.2: XMI representation of the code list *AdministrativeHierarchyLevel* based on the proposed formal definition of the stereotype «CodeList»

```
<packagedElement xmi:type="uml:Enumeration" xmi:id="ic3EeGfa4oH" name="AdministrativeHierarchyLevel">
  <ownedLiteral xmi:type="uml:EnumerationLiteral" name="1stOrder"/>
  <ownedLiteral xmi:type="uml:EnumerationLiteral" name="2ndOrder"/>
  <ownedLiteral xmi:type="uml:EnumerationLiteral" name="3rdOrder"/>
  <ownedLiteral xmi:type="uml:EnumerationLiteral" name="4thOrder"/>
  <ownedLiteral xmi:type="uml:EnumerationLiteral" name="5thOrder"/>
  <ownedLiteral xmi:type="uml:EnumerationLiteral" name="6thOrder"/>
</packagedElement>
...
<thecustomprofile:CodeList base_Enumeration="ic3EeGfa4oH"/>
```

The standard ISO/TC 19103 states that enumerations should only be used when the value set is fixed. Otherwise, code lists should be used, since the value set of code lists can be extended during system runtime. This raises the question what exactly the term 'extended' means. Does it mean that further values can be added to the predefined values in an instance of the code list? Or could it also mean that an arbitrary value different from the values predefined in the code list can simply be used as attribute value? Also the term 'during system runtime' is not fully clear. What exactly does a code list look like at run-time? Is it represented by an UML object diagram, by a data transfer format or by byte code, which at run-time is translated into machine language by a virtual machine?

Furthermore, can this behaviour really be achieved by modelling a code list using the metamodel element *Class*? From a UML (and generally object-oriented) point of view, extending a class by new properties means that a subclass needs to be created to which these new properties are then added[2]. From a programming language point of view it depends on the programming language, whether classes can be extended by further attributes during run-time, since only dynamic programming languages provide this kind of functionality. From a data transfer format point of view it depends on the encoding rules used; they define by which data structure a code list has to be represented within the data transfer format to remain extensible.

These questions should be clarified first, before a decision is made how to represent a code list in UML. However, for the purpose of using UML for modelling geographic information and generating data transfer formats from these UML models, as is one of the major applications of the standard ISO/TS 19103 currently (e. g. in the context of the standards ISO 19118 and ISO 19136 Annex E in general and the current SDI initiative INSPIRE in particular), code lists can be represented without any problems using the proposed approach. It is then a matter of the encoding rule to choose a suitable extensible data structure the code list is represented in within the data transfer format. Furthermore, also the proposed form of representing the key-value pairs as *value=key* does not pose a problem to software tools to distinguish the key from the value, as they are separated from each other by the equals sign.

5.1.2.2 Formal definition of the stereotype «Union»

The discussion of the stereotype «Union» in section 5.1.1.2, page 79, showed that the current definition of the stereotype causes a semantic modification of the UML metamodel. Furthermore, it was argued that no reason exists for a union to possess an identity. It is therefore proposed to define the stereotype «Union» as an extension of the UML metaclass *DataType*, as is shown in figure 5.3.

To fully preserve the semantics of the concept *Union*, the OCL constraint in listing 5.3 is defined. The first invariant of the OCL constraint states that data types to which the stereotype «Union» is applied must own at least two properties. The second invariant defines that for each property the lower bound of the multiplicity must be 0 and the upper bound must be 1. The third invariant defines that the data types must exhibit an OCL constraint themselves. This OCL constraint should then specify by means of the Boolean operator *XOR* that instances of this union are allowed to exhibit a value for only one of the properties defined. This third invariant is indeed not very precise; however, at the meta level the concrete instances are not yet known and, thus, also not the properties and the number of properties they define.

Listing 5.3: OCL constraint of the proposed formally defined stereotype «Union»

```
context Union
    inv: self.base_DataType.ownedAttribute->size()>=2
    inv: self.base_DataType.ownedAttribute->forAll(c | c.lower=0 and c.upper=1)
    inv: self.base_DataType.ownedRule->notEmpty()
```

This is illustrated by using the union *AA_UUID* from section 5.1.1.2, page 79, again. Figure 5.5 shows the new union. It is now modelled as a data type to which the stereotype «Union» has been applied; thus, the union exhibits now, on the one hand, the keyword «datatype» and, on the other hand,

[2]Extension by subclassing is not the only way to add new properties to a class. A class can also be extended by superclassing, i.e. by adding the properties to a superclass from where they are inherited by the class to extend.

the stereotype «Union». Furthermore, the properties have been assigned the multiplicity [0..1]. Above that, an OCL constraint specifies by means of the Boolean operator *XOR* that in an instance of the union either the property *UUID* or the property *UUIDundZeit* is allowed to have a value.

However, specifying the OCL constraint in this way only results in true, when the union defines an even number of properties. Figure 5.6 shows a union which defines three properties and, thus, requires specifying an OCL constraint in the form of *punkt->notEmpty() xor linie->notEmpty() xor flaeche->notEmpty()*. According to Boolean algebra, this OCL constraint would not only result in true, when exactly one of the properties has a value, but also when all three properties exhibit values. By extending the OCL constraint as demonstrated in figure 5.6, the latter case can explicitly be avoided from becoming true. This means, whenever a union contains an odd number of attributes, the OCL constraint needs to be defined in this extended form.

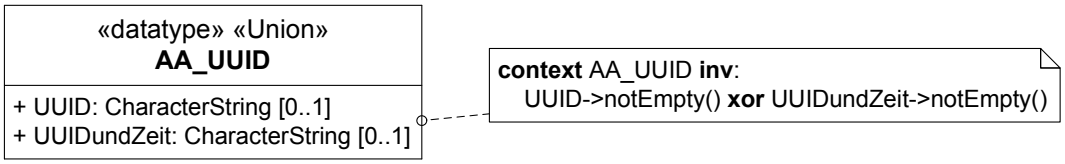


Figure 5.5: Union *AA_UUID* from the AAA reference model (AdV 2009) based on the proposed formal definition of the stereotype «Union». The OCL constraint shows the use of the XOR operator when the union defines an even number of properties.

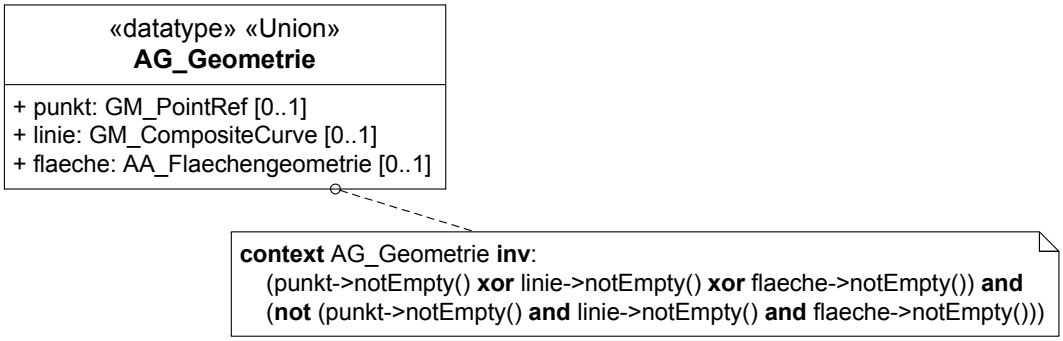


Figure 5.6: Union *AG_Geometrie* from the AAA reference model (AdV 2009) based on the proposed formal definition of the stereotype «Union». The OCL constraint shows the use of the XOR operator when the union defines an odd number of properties.

5.1.2.3 Formal definition of the stereotype «Leaf»

The stereotype «Leaf» is applied to UML packages which do not contain any other UML packages. The definition provided by the standard ISO/TS 19103 is non-ambiguous. The stereotype «Leaf» constrains the UML metaclass *Package* in such a way that its semantics remains consistent with the semantics of the UML metaclass *Package*. However, this restriction should be defined formally as well, which is achieved by specifying the OCL constraint in listing 5.4:

Listing 5.4: OCL constraint of the proposed formally defined stereotype «Leaf»

```
context Leaf inv:
    self.base_Package.nestedPackage->size()=0
```

5.1.3 Current revision of the standard ISO/TS 19103

The standard ISO/TS 19103 is currently under revision. A new Draft International Standard (ISO/DIS 19103) was circulated among the ISO member bodies from October 2013 to early January 2014 (ISO 2013a). According to this DIS, the revised standard will be based on UML version 2.4.1 and on OCL version 2.3.1, which are defined by the standards ISO/IEC 19505-2 and ISO/IEC 19507, respectively.

In the DIS a formal UML profile is provided which models all stereotypes defined in the DIS as extensions of appropriate UML metamodel elements (cf. figure C.3, page 184). In this UML profile the stereotypes «CodeList» and «Union» extend the metaclasses *CodeList* and *Union*, respectively, which are themselves modelled as subclasses of the metaclass *DataType*. However, the metaclasses *CodeList* and *Union* do not exist within the UML metamodel as defined by the standard ISO/IEC 19505-2, which is why this representation rather defines another modification of the existing UML metamodel, instead of a UML profile compliant with the UML profile definition of the OMG.

Furthermore, the DIS does not explain why the stereotypes «Enumeration» and «Interface» are defined, even though corresponding metamodel elements already exist in the UML metamodel. The same applies to the stereotype «Type» which is already predefined by the UML metamodel as part of the StandardProfileL2. Defining stereotypes would be justified if specific tag definitions shall be added to UML models, but the formal UML profile does not define any tag definitions for the above mentioned stereotypes.

The DIS defines rules for mapping UML models based on the standard ISO/TS 19103 and on UML 1 to UML models based on the revision and on UML 2. By defining the stereotypes «CodeList» and «Union» as proposed in the sections 5.1.2.1, page 82, and 5.1.2.2, page 83, respectively, mapping rules could be defined likewise for mapping UML models based on the standard ISO/TS 19103 to UML models based on the proposed formal UML profile defined above.

5.2 The UML profile of the standard ISO 19109

Another important basis for UML profiles in geographic information is the standard ISO 19109 which was already introduced in section 2.3.2, page 14, and which defines rules for the development of application schemas which are based on the General Feature Model (GFM). The fundamental concepts defined in the GFM are *application schema* and *feature type*. However, the standard does not define any stereotypes for these concepts; instead, the corresponding stereotypes are defined in the standard ISO 19136 (cf. section 5.3, page 86).

The standard ISO 19109 is currently under revision. A new Draft International Standard (ISO/DIS 19109) was circulated among the ISO member bodies from October 2013 to March 2014 (ISO 2013b). The DIS defines a UML profile which is to be used as modelling language for modelling application schemas. Similar to the UML profile defined by the standard ISO/TS 19103 (cf. section 5.1.1, page 76) also this UML profile goes far beyond the scope of what the UML profile definition of the OMG defines as a UML profile. The UML profile in ISO/DIS 19109 contains not only stereotypes, but also data types from ISO/DIS 19103, some further requirements on UML associations as well as

recommendations on attributes, operations and roles (ISO 2013b). According to this DIS, the revised standard will be based on UML version 2.4.1.

The DIS defines in particular the stereotypes «ApplicationSchema», «FeatureType» and «estimated» which complement the stereotypes «Leaf», «Datatype», «Enumeration», «CodeList» and «Union» from ISO/DIS 19103; all these stereotypes are part of the modelling language for modelling application schemas. The stereotypes are provided in tabular form only together with further definitions within the text. Based on this information, a formal UML profile diagram was created for this thesis (cf. figure C.4, page 184). It can be noted that the stereotypes «FeatureType», «Datatype», «Enumeration», «CodeList» and «Union» are now defined as extensions of the metaclass *Classifier* which means that any of these stereotypes can be applied to any Classifier. The stereotype «FeatureType», for instance, could now be applied to the metaclasses *Class*, *Enumeration*, *DataType* and even *PrimitiveType* since all these metaclasses are subclasses of the metaclass *Classifier* – presumably, this is not the intended way of applying the stereotypes. Furthermore, the stereotype «estimated» extends the metaclasses *Attribute* and *AssociationRole*. They, however, do not exist anymore in UML 2. The correct metaclass for stereotypes which are to be applied to attributes and associations is now the metaclass *Property*.

5.3 The UML profile of the standard ISO 19136

Within the ISO 191xx series of geographic information standards another UML profile is defined by the standard *ISO 19136:2007 Geoinformation — Geography Markup Language (GML)* (ISO 2007). The standard ISO 19136 defines on the one hand GML, an XML-based language for modelling, transferring and storing geospatial information; on the other hand, Annex E of this standard specifies an encoding rule for mapping UML application schemas to GML application schemas.

The encoding rule requires that UML application schemas are modelled compliant with the standards ISO/TS 19103 and ISO 19109, which implies that the stereotypes from these standards have to be applied to each UML application schema for which GML application schemas are to be generated. These stereotypes are complemented in the encoding rule by additional stereotypes and by tag definitions to enable an automatic mapping of UML application schemas to GML application schemas. Thus, the specifications given in Annex E of the standard ISO 19136 form a UML profile of its own which will be referred to in the following as *ISO 19136 UML profile*.

The encoding rule is provided in textual and tabular form. Based on this information, table 5.2 was created which intends to give an overview of the stereotypes required by the encoding rule, the UML metaclasses they extend and from which standard they originate. Furthermore, a formal UML profile diagram was created for this thesis based on the information provided in ISO 19136 Annex E (cf. figure C.5, page 185).

The following sections will discuss the stereotypes in more detail, point out the deficits of the UML profile defined by the standard ISO 19136 and finally propose a formal definition of the stereotypes as part of a UML profile compliant with the UML profile definition of the OMG.

5.3.1 Discussion of the ISO 19136 stereotypes

The discussion includes for each stereotype a table which lists the tag definitions, their types and multiplicities as well as their intended use. These tables are provided, as the stereotypes can only be analysed in their entirety when also the tag definitions defined for them are taken into account in the analysis. Furthermore, the tables are to make more explicit the types, multiplicities and intended

Table 5.2: Stereotypes of the ISO 19136 UML profile

Stereotype	UML metaclass	Origin of stereotype
«Application Schema»	Package	ISO 19136 Annex E
any or no stereotype	Package	ISO 19136 Annex E
«Leaf»	Package	ISO/TS 19103
«Import»	Dependency	UML 1
«FeatureType»	Class	ISO 19136 Annex E
no stereotype	Class	ISO 19136 Annex E
«Type»	Class	UML 1
«Enumeration»	Class	UML 1
«CodeList»	Class	ISO/TS 19103
«Union»	Class	ISO/TS 19103 / ISO 19107
«DataType»	Class	UML 1
«Association»	AssociationEnd	UML 1
no stereotype	Attribute, AssociationEnd	ISO 19136 Annex E

use of the tag definitions. Currently, the standard does not provide such precise definitions for them, which is why the information in the tables is sometimes rather deduced from the descriptions given in the document.

5.3.1.1 The stereotypes «Application Schema» and «Leaf» and UML packages with any or no stereotype

As mentioned in section 5.2, page 85, the concept of the application schema has already been introduced in the standard ISO 19109, however, the corresponding stereotype «Application Schema» which represents this concept is defined in the standard ISO 19136 Annex E. The following definition is given there:

> The UML Application Schema shall be represented by a package with the stereotype <<Application Schema>>. This package shall contain (i.e. own directly or indirectly) all UML model elements to be mapped to object types in the GML application schema. The package may include other packages without the stereotype <<Application Schema>> to group the different UML model elements within the application schema. (ISO 2007)

According to this definition one has to differentiate between two types of UML packages: UML packages which represent application schemas and to which, thus, the stereotype «Application Schema» is applied, and UML packages which are contained in the former packages and to which any stereotype except the stereotype «Application Schema» can be assigned, including the stereotype «Leaf» from the standard ISO/TS 19103, or are without stereotype at all.

Table 5.3 lists the tag definitions which are defined in the standard ISO 19136 Annex E for the stereotype «Application Schema» and table 5.4 lists the tag definitions for UML packages contained within UML application schemas. Furthermore, the stereotype «Leaf» which was defined in the standard ISO 19103 without tag definitions, is according to the standard ISO 19136 Annex E complemented by the tag definitions from table 5.4 (cf. figure C.5, page 185).

The stereotypes «Application Schema» and «Leaf» as well as the related tag definitions can be defined formally as part of a UML profile compliant with the UML profile definition of the OMG. However, the tag definitions for UML packages contained within UML application schemas can only be used as is when creating UML 1 compliant UML application schemas, since they contradict the UML 2 profile definition of the OMG, according to which tag definitions can only be applied to a UML model element in connection with a stereotype (cf. section 2.4.1, page 19). Thus, in the context of UML 2, a suitable stereotype needs to be defined together with the tag definitions as part of the ISO 19136 UML profile. This stereotype can then be applied to UML packages without stereotype, but also to UML packages which already possess an arbitrary stereotype except «Application Schema». In the ISO 19136 UML profile diagram in figure C.5, page 185, the extension was modelled without stereotype name to reflect the current situation and also, because the standard ISO 19136 Annex E currently conforms to the standard ISO/TS 19103 which, in turn, is based on UML 1.4.2.

Table 5.3: ISO 19136 UML profile: Tag definitions of the stereotype «Application Schema»

Tag definition	Meaning	Type	Multiplicity	Use
documentation	Description of the application schema	String	0..1	Conceptual level, GML encoding
xsdDocument	Name of the XML schema document to be generated	String	1	GML encoding
targetNamespace	URI of the XML target namespace	String	1	GML encoding
xmlns	Prefix of the XML target namespace	String	1	GML encoding
version	Version number of the application schema (if no value is specified, 'unknown' is to be used)	String	1	Conceptual level, GML encoding
gmlProfileSchema	URL of a possible GML profile	String	0..1	GML encoding

Table 5.4: ISO 19136 UML profile: Tag definitions of UML packages contained within UML application schemas

Tagged value	Meaning	Type	Multiplicity	Use
documentation	Description of the package	String	0..1	Conceptual level, GML encoding
xsdDocument	Name of the XML schema document to be generated	String	0..1	GML encoding

5.3.1.2 The stereotype «Import»

The standard ISO 19136 Annex E defines the use of the stereotype «Import» as follows:

> Dependencies between packages shall be modelled explicitly. Permission elements with stereotype <<import>> or unspecified dependency elements between packages shall be used to express the dependency of elements in a package from elements in another package. (ISO 2007)

This means, whenever in a UML model shall be expressed that UML elements in one UML package are dependent from UML elements in another UML package, either a general UML dependency

relationship which is not further specified by a stereotype or an import relationship, which is specified by the stereotype «import», is to be used between these UML packages.

This definition is only appropriate with respect to UML 1. In UML 1 a UML metamodel element *Dependency* exists, which predefines several kinds of dependency and corresponding keywords. One of these predefined keywords is «import». Since the standard ISO 19136 Annex E conforms to the standard ISO/TS 19103, and ISO/TS 19103, in turn, is based on UML 1.4.2, and since above definition refers to «import» as a stereotype, the extension was modelled accordingly in the ISO 19136 UML profile diagram in figure C.5, page 185. Actually, «import» is not a stereotype, but a keyword. Since no tag definitions are defined in the standard ISO 19136 Annex E for «import», no necessity exists to define an additional stereotype providing tag definitions for this keyword. For the same reason, no problems in deriving GML application schemas should occur when using «import» as keyword in the modelling of UML application schemas.

In UML 2, import relationships between packages are defined by the UML metamodel element *PackageImport* and the keywords «import» for public package imports and «access» for private package imports (ISO 2012b). A UML metamodel element *Dependency* exists as well, but it does not express import relationships between packages anymore.

5.3.1.3 The stereotype «FeatureType» and UML classes without stereotype representing object types

As mentioned in section 5.2, page 85, the concept of the feature type has already been introduced in the standard ISO 19109. The corresponding stereotype «FeatureType», however, which represents this concept is defined in the standard ISO 19136 Annex E. Furthermore, a distinct object type is introduced in ISO 19136 Annex E. The following definitions are given:

> Feature types shall be modelled as UML classes with stereotype <<FeatureType>> [...]. (ISO 2007)

> Object types shall be modelled as UML classes with no stereotype. Object types are types where the instances shall have an identity, but which are not feature types [...]. (ISO 2007)

The tag definitions defined in the standard ISO 19136 Annex E for the stereotype «FeatureType» and for UML classes representing object types are listed in table 5.5. Based on the above definitions, the stereotype «FeatureType» and object types extend the UML metaclass *Class*. The stereotype «FeatureType» as well as the related tag definitions can be defined formally as part of a UML profile compliant with the UML profile definition of the OMG. For object types, however, the same conclusion can be drawn as for UML packages contained within UML application schemas (cf. section 5.3.1.1, page 87). The tag definitions for object types can only be used as is, when creating UML 1 compliant UML application schemas. In the context of UML 2, a stereotype has to be defined together with the required tag definitions, otherwise the tag definitions cannot be applied to UML model elements.

5.3.1.4 The stereotype «Type»

The standard ISO 19136 Annex E gives the following definition for using the stereotype «Type»:

> UML classes with stereotype <<Type>> may have zero or more operations (these are not mapped to the GML application schema), attributes or associations. (ISO 2007)

Table 5.5: ISO 19136 UML profile: Tag definitions of the stereotype «FeatureType» and of UML classes representing object types

Name	Meaning	Type	Multiplicity	Use
documentation	Description of the feature type / object type	String	0..1	Conceptual level, GML encoding
noPropertyType	An XML complex type is to be generated carrying the name of the feature type / object type with the suffix PropertyType	Boolean	1	GML encoding
byValuePropertyType	An XML complex type is to be generated carrying the name of the feature type / object type with the suffix PropertyByValueType	Boolean	1	GML encoding
isCollection	The feature type / object type is to be encoded as object collection	Boolean	1	GML encoding

Table 5.6 lists the tag definitions defined in the standard ISO 19136 Annex E for the stereotype «Type». The semantics of this stereotype is equivalent to the semantics of the stereotype «Type» predefined in the UML 1 specification and in the UML 2 specification as part of the UML StandardProfileL2, thus, ISO 19136 Annex E actually only extends the predefined stereotype by tag definitions. This fact will be elaborated further in section 5.3.2.3, page 98. For the moment, the stereotype is displayed in the ISO 19136 UML profile diagram in figure C.5, page 185, as understood by ISO 19136 Annex E, i. e. as extension of the UML metaclass *Class*.

Table 5.6: ISO 19136 UML profile: Tag definitions of the stereotype «Type»

Name	Meaning	Type	Multiplicity	Use
documentation	Description of the type	String	0..1	Conceptual level, GML encoding
noPropertyType	An XML complex type is to be generated carrying the name of the type with the suffix PropertyType	Boolean	1	GML encoding
byValuePropertyType	An XML complex type is to be generated carrying the name of the type with the suffix PropertyByValueType	Boolean	1	GML encoding
isCollection	The type is to be encoded as object collection	Boolean	1	GML encoding
xmlSchemaType	The XML schema type of a corresponding canonical XML schema encoding	String	0..1	GML encoding

5.3.1.5 The stereotypes «Enumeration» and «DataType»

The stereotypes «Enumeration» and «DataType» have already been discussed in detail in the context of the ISO/TS 19103 UML profile (cf. section 5.1.1.3, page 80). The standard ISO 19136 Annex E defines the modelling of the stereotypes «Enumeration» and «DataType» as follows:

> Enumerations shall be modelled as UML classes with stereotype <<Enumeration>>. (ISO 2007)

> All other data types [i. e. data types which are not feature types, object types, types, enumerations, code lists or union types] shall be modelled as UML classes with stereotype <<DataType>>. (ISO 2007)

The tag definitions which are defined in the standard ISO 19136 Annex E for the stereotypes «Enumeration» and «DataType» are listed in tables 5.7 and 5.8, respectively. As explained in section 2.4.2, page 22, and also in the context of the ISO/TS 19103 UML profile, «Enumeration» and «DataType» are not stereotypes, but keywords denoting the UML metamodel elements *Enumeration* and *DataType*, respectively. Thus, they have to be treated differently regarding their formal definition in combination with tag definitions as part of a UML profile compliant with the UML 2 profile definition of the OMG. This fact will be elaborated further in section 5.3.2.2, page 97. For the moment, the stereotypes are displayed in the ISO 19136 UML profile diagram in figure C.5, page 185, as understood by ISO 19136 Annex E, i. e. as extensions of the UML metaclass *Class*.

Table 5.7: ISO 19136 UML profile: Tag definitions of the stereotype «Enumeration»

Name	Meaning	Type	Multiplicity	Use
documentation	Description of the enumeration	String	0..1	Conceptual level, GML encoding

Table 5.8: ISO 19136 UML profile: Tag definitions of the stereotypes «DataType» and «Union»

Name	Meaning	Type	Multiplicity	Use
documentation	Description of the data type / union	String	0..1	Conceptual level, GML encoding
noPropertyType	An XML complex type is to be generated carrying the name of the data type / union with the suffix PropertyType	Boolean	1	GML encoding

5.3.1.6 The stereotypes «CodeList» and «Union»

The stereotypes «CodeList» and «Union» as well as the semantic modification they impose on the UML metamodel have already been discussed in detail in the context of the ISO/TS 19103 UML profile (cf. sections 5.1.1.1, page 77, and 5.1.1.2, page 79, respectively). Regarding the modelling of the stereotypes «CodeList» and «Union», the standard ISO 19136 Annex E specifically defines that:

> Code lists shall be modelled as UML classes with stereotype <<CodeList>>. (ISO 2007)

> Union types shall be modelled as UML classes with stereotype <<Union>> [...]. (ISO 2007)

Table 5.9 lists the tag definitions which are defined in the standard ISO 19136 Annex E for the stereotype «CodeList». The tag definitions defined for the stereotype «Union» are identical to those defined for the stereotype «DataType» in table 5.8, therefore, they are not listed again. According to the above definitions, the stereotypes «CodeList» and «Union» are to be defined as extensions of the metaclass *Class*. In the context of the proposed formal ISO/TS 19103 UML profile, approaches have already been introduced for how to define these stereotypes without causing a semantic modification of the UML metamodel (cf. section 5.1.2.1, page 82, and 5.1.2.2, page 83, respectively). These approaches will be elaborated further in section 5.3.2.3, page 98, taking into account the tag definitions required for the stereotypes «CodeList» and «Union» in the context of the ISO 19136 UML profile. For the moment, the stereotypes are displayed in the ISO 19136 UML profile diagram in figure C.5, page 185, as understood by ISO 19136 Annex E, i. e. as extensions of the UML metaclass *Class*.

Table 5.9: ISO 19136 UML profile: Tag definitions of the stereotype «CodeList»

Name	Meaning	Type	Multiplicity	Use
documentation	Description of the code list	String	0..1	Conceptual level, GML encoding
asDictionary	The code list is to be encoded as dictionary	Boolean	1	GML encoding

5.3.1.7 UML attributes and UML association ends

The standard ISO 19136 Annex E defines the modelling of UML association ends as follows:

> Every UML association shall be an association with exactly two association ends. Both association ends shall connect to a feature, object or data type and shall have no stereotype or the stereotype «association». (ISO 2007)

No stereotype is defined in the standard for UML attributes. Table 5.10 lists the tag definitions which are defined in the standard ISO 19136 Annex E for UML attributes and UML association ends. The stereotype «Association» as well as the related tag definitions can be defined formally as part of a UML profile compliant with the UML profile definition of the OMG. However, for UML attributes and UML association ends without stereotype the same conclusion can be drawn as for UML packages contained within UML application schemas in section 5.3.1.1, page 87, and object types in section 5.3.1.3, page 89. According to the UML profile definition of the OMG, tag definitions can only be applied to a UML model element in connection with a stereotype (cf. section 2.4.1, page 19). Thus, first a stereotype has to be defined together with the required tag definitions, otherwise the tag definitions cannot be used.

The standard ISO 19136 Annex E, however, does not give a precise answer on the question of which UML metaclasses the stereotype for UML attributes and UML association ends is to be defined as an extension. For UML attributes, a UML metamodel element *Attribute* exists in UML 1 of which the stereotype can be extended. In UML 2, however, the concept was replaced by the UML metamodel element *Property*. Regarding UML association ends, the standard ISO 19136 Annex E alternately

Table 5.10: ISO 19136 UML profile: Tag definitions of UML attributes and UML association ends

Name	Meaning	Type	Multiplicity	Use
documentation	Description of the attribute or association end	String	0..1	Conceptual level, GML encoding
sequenceNumber	Ordering of attributes and association roles in the XML schema	Integer	1	Conceptual level, GML encoding
inlineOrByReference	Encoding of the property value type	String	0..1	GML encoding
isMetadata	Attribute or association end is metadata property	Boolean	1	GML encoding

speaks of *association end* and *association role*. In the *UML Profile for GML Applications Schemas* (cf. appendix C.1, page 179) the UML metaclass *AssociationRole* is used for defining an appropriate stereotype. However, this metaclass exists only in UML 1 and was defined there as part of the *Collaborations* package. Collaborations describe which elements of a system have to communicate with each other to achieve a certain task:

> In the metamodel, an AssociationRole specifies a restricted view of an Association used in a Collaboration. An AssociationRole is a composition of a set of AssociationEndRoles corresponding to the AssociationEnds of its base Association. (ISO 2005b)

In UML 2 the concept is no longer available:

> The contents of a collaboration is specified as its internal structure relying on roles and connectors; the concepts of ClassifierRole, AssociationRole, and AssociationEndRole have been superseded. (ISO 2012b)

Furthermore, in UML 1 the metamodel element *AssociationEnd* exists:

> In the metamodel, an AssociationEnd is part of an Association and specifies the connection of an Association to a Classifier. (ISO 2005b)

In UML 2, however, the metaclass is no longer available, an association end is now represented by the metamodel element *Property*:

> When a property is owned by a classifier other than an association via ownedAttribute, then it represents an *attribute* of the class or data type. When related to an association via memberEnd or one of its specializations, it represents an end of the association. (ISO 2012b)

> AssociationEnd was a metaclass in prior UML, now demoted to a member of Association. (ISO 2012b)

Based on these definitions, *AssociationRole* does not seem to be the correct UML metaclass of which to extend a stereotype for UML association ends. Since the standard ISO 19136 Annex E conforms to ISO/TS 19103, and ISO/TS 19103, in turn, is based on UML 1.4.2, the UML metamodel element *AssociationEnd* was chosen in the ISO 19136 UML profile diagram in figure C.5, page 185, as the most appropriate UML metaclass.

5.3.1.8 Conclusion

When comparing the UML profiles defined by the standards ISO/TS 19103 and ISO 19136 Annex E, it can be noticed that the UML profile defined by ISO 19136 Annex E represents an extended version of the UML profile defined by ISO/TS 19103 as it complements the ISO/TS 19103 UML profile by additional stereotypes and provides tag definitions for most of the stereotypes. Furthermore, when reading the explanations of each tag definition, it becomes apparent that the tag definitions defined in ISO 19136 Annex E are mainly required for providing additional information enabling an automatic mapping of the individual UML model elements in UML application schemas to XML schema elements in GML application schemas, but not for enhancing UML application schemas at the conceptual level.

The indication *no stereotype* in table 5.2, page 87, refers to extensions regarding the UML metaclasses *Package*, *Class*, *Attribute* and *AssociationEnd* (UML 1) / *Property* (UML 2) for which the standard ISO 19136 Annex E defines tag definitions, but no stereotypes. These tag definitions have to be applied to UML application schemas for being able to derive correct GML application schemas automatically. They can be used as is, when creating UML 1 compliant UML application schemas, however, they contradict the UML 2 profile definition of the OMG, according to which tag definitions must occur in combination with a named stereotype. Similarly, this also holds true for the indication *any stereotype* in table 5.2. Therefore, to be able to apply the specified tag definitions from ISO 19136 Annex E in UML 2 compliant application schemas, first a stereotype has to be defined together with the required tag definitions, otherwise they cannot be made use of.

«Enumeration» and «DataType» are defined in the UML metamodel as keywords for the corresponding UML metamodel concepts *Enumeration* and *DataType*. The standard ISO 19136 Annex E does not consider this fact, but rather treats them as stereotypes extending the UML metamodel concept *Class* and, in addition, specifies tag definitions for them. A solution needs to be found which takes into account the tag definitions and which at the same time preserves the semantics of the UML metamodel concepts. Similarly, «Import» represents a keyword in the UML metamodel and a stereotype in ISO 19136 Annex E. However, since no tag definitions are specified for this keyword, its use does not pose a problem.

Regarding the stereotypes «CodeList» and «Union», the same conclusion applies as was already given in the context of the ISO/TS 19103 UML profile (cf. section 5.1.1.4, page 80), i. e. their semantics causes a semantic modification of the UML metamodel.

The stereotypes «Leaf» and «Type» actually represent extended versions of already existing stereotypes, nevertheless they can be defined formally as part of a UML profile compliant with the UML profile definition of the OMG. Also the stereotypes «Application Schema», «FeatureType» and «Association» can be defined formally according to the UML profile definition of the OMG and do not cause any problems in use.

5.3.2 Proposed formal definition of an ISO 19136 UML profile

In section 2.4.3, page 23, the UML package merge concept was introduced which enables the contents of two UML packages to be combined into one UML package. Since UML profiles are UML packages themselves, package merge can be regarded as a convenient concept for creating UML profiles targeted at different communities within a certain domain. In this way, in a first step base definitions relevant to all communities within a domain can be defined which afterwards can be extended by specific concepts required only within a specific community.

For the geospatial domain, the ISO/TS 19103 UML profile proposed in section 5.1.2, page 81, can represent such a base definition, since the stereotypes it provides are relevant to all communities modelling geographic information. The ISO 19136 UML profile makes use of this UML profile as well, but it extends the existing stereotypes by tag definitions and also defines additional stereotypes required for being able to automatically derive GML application schemas from UML application schemas. Thus, in terms of the UML package merge concept, the ISO/TS 19103 UML profile represents the merged package and the ISO 19136 UML profile represents the receiving package as is displayed in figure 5.7. In addition, also the UML StandardProfileL2 defined in the UML 2 Superstructure has to be included into the merge, since it contains the stereotype «Type» for which the standard ISO 19136 Annex E defines specific tag definitions.

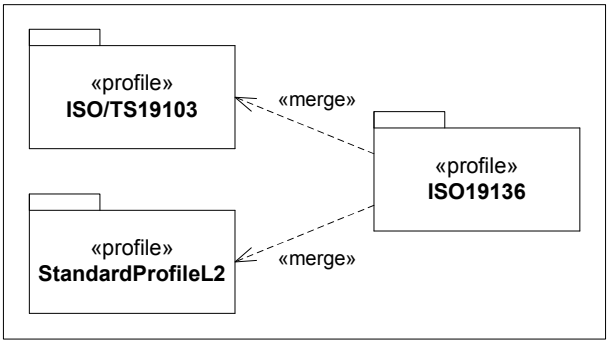


Figure 5.7: Merging of the ISO/TS 19103 UML profile and of the UML StandardProfileL2 into the proposed formal ISO 19136 UML profile

The formal ISO 19136 UML profile proposed in this section takes into account the UML package merge concept. Figure 5.8 displays the proposed formal ISO 19136 UML profile. Stereotypes which have already been defined in the proposed formal ISO/TS 19103 UML profile in section 5.1.2, page 81, are not defined anew, but only complement the base definitions of the standard ISO/TS 19103 by additional concepts defined in the standard ISO 19136 Annex E. To be compliant with the UML 2 profile definition of the OMG, specific stereotypes are defined for all metaclass extensions indicated in the above discussion by *no stereotype* (cf. table 5.2, page 87) as well as for the keywords to allow for using the required tag definitions in modelling UML 2 compliant application schemas. All stereotypes defined formally in the proposed formal ISO 19136 UML profile are discussed in more detail in the following sections.

Section 5.3.1.2, page 88, pointed out that «import» is not a stereotype, but a keyword. Since no tag definitions are defined in the standard ISO 19136 Annex E, no necessity exists to define a specific stereotype providing tag definitions for this keyword; therefore, «import» has not been included in the proposed formal ISO 19136 UML profile. Also, the stereotype «Association» has not been included in the formal ISO 19136 UML profile. On the one hand, the standard ISO 19136 Annex E does not mandatorily prescribe its use in modelling UML application schemas (cf. section 5.3.1.7, page 92). On the other hand, a stereotype has to be defined anyway for UML attributes to make use of the tag definitions defined in ISO 19136 Annex E. Since UML attributes as well as UML association ends are represented in UML 2 by the metamodel element *Property*, and since both require them same tag definitions, too, the stereotype defined for UML attributes can be used for UML association ends as well.

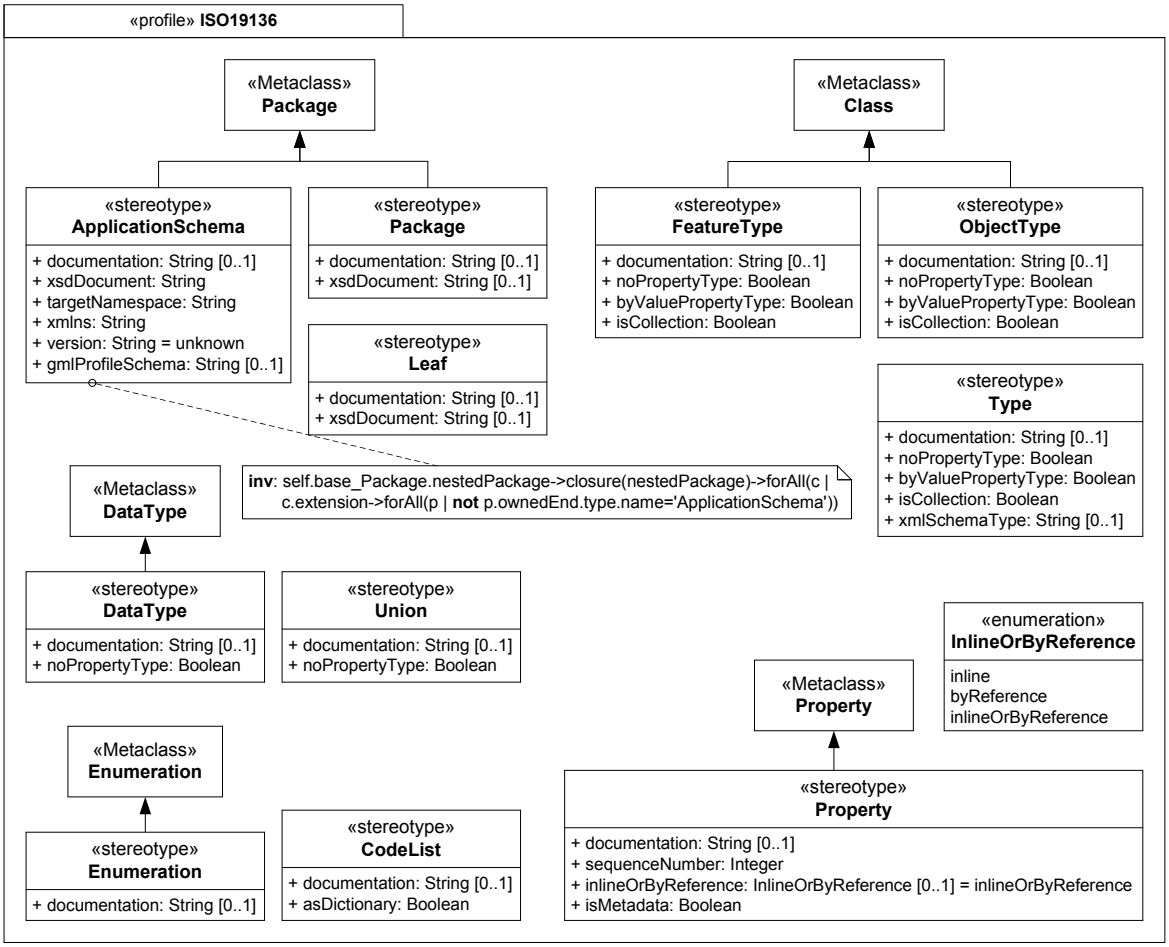


Figure 5.8: Proposed formal ISO 19136 UML profile

5.3.2.1 Formal definition of the stereotypes «ApplicationSchema» and «FeatureType»

As discussed in the sections 5.3.1.1, page 87, and 5.3.1.3, page 89, the stereotypes «Application Schema» and «FeatureType» as well as their related tag definitions can be defined formally according to the UML 2 profile definition of the OMG; thus, no modifications to the existing definitions are required.

However, the name of the stereotype «Application Schema» is changed to «ApplicationSchema» in the proposed formal ISO 19136 UML profile, since all other stereotype names composed of two words are written without space between the words as well. Furthermore, several other UML profiles, such as the INSPIRE UML profile (cf. section 5.4, page 100) or the AAA UML profile (cf. section 5.5.1, page 108), define a stereotype with the name «ApplicationSchema». According to the explanations in section 2.4.1, page 19, «Application Schema» and «ApplicationSchema» are two different concrete syntax elements, each of them representing a different abstract syntax element. Therefore, keeping the stereotype name as is, i. e. with a space between the two words, but defining the same stereotype

in other UML profiles without space, would result in two different stereotypes – this does not seem intended.

In addition, the OCL constraint in listing 5.5 is added to the stereotype «ApplicationSchema», restraining UML application schema packages from containing other UML packages with the stereotype «ApplicationSchema». This is equivalent to the textual constraint 'Not nested in another applicationSchema package' (ISO 2013b) used in the UML profile of ISO/DIS 19109 (cf. figure C.4, page 184).

Listing 5.5: OCL constraint of the proposed formally defined stereotype «ApplicationSchema»

```
context ApplicationSchema inv:
    self.base_Package.nestedPackage->closure(nestedPackage)->forAll(c | c.extension->forAll(p | not
        p.ownedEnd.type.name='ApplicationSchema'))
```

5.3.2.2 Formal definition of stereotypes for the keywords «enumeration» and «dataType»

As explained in section 2.4.2, page 22, and also in the context of the ISO/TS 19103 UML profile in section 5.1, page 76, «Enumeration» and «DataType» are not stereotypes, but keywords denoting the UML metamodel elements *Enumeration* and *DataType*, respectively. Thus, they have to be treated differently regarding their formal definition in combination with tag definitions as part of a UML profile compliant with the UML profile definition of the OMG. Appropriate stereotypes need to be defined which extend the metaclasses *Enumeration* and *DataType*, respectively. The question arises, which stereotype names to choose, since naming a stereotype identically to an existing UML keyword can lead to confusion and, therefore, is discouraged from by the UML specification (ISO 2012b).

However, the UML specification also recommends to start keyword names with a small letter and stereotype names with a capital letter which makes the difference visible at first view, provided this rule is followed strictly. Based on this argument, the decision was made to name the stereotypes in the proposed formal ISO 19136 UML profile identically to the already existing keywords and to strictly adhere to the upper and lower case rule. This means, a stereotype «Enumeration» is defined which extends the UML metaclass *Enumeration* and a stereotype «DataType» which extends the UML metaclass *DataType*. Both stereotypes are, in addition, provided with the required tag definitions. When creating now a UML application schema based on the proposed formal ISO 19136 UML profile, UML model elements which represent on the one hand a data type (expressed by the keyword «dataType») and to which on the other hand the tag definitions relevant for data types are applied (expressed by the stereotype «DataType») will appear in the UML application schema as illustrated in figure 5.9, i. e. keyword and stereotype names are displayed next to each other above the name of the UML model element.

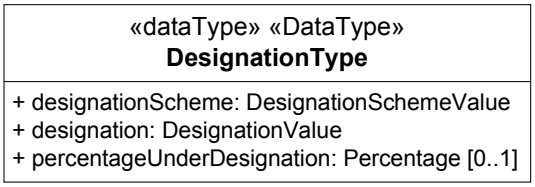


Figure 5.9: Data type *DesignationType* from the application schema INSPIRE Protected Sites Simple (JRC 2014f) with formally proposed stereotype «DataType» applied

5.3.2.3 Formal definition of the stereotypes «CodeList», «Union», «Leaf» and «Type»

The stereotypes «CodeList», «Union» and «Leaf» are already defined in the proposed formal ISO/TS 19103 UML profile in section 5.1.2, page 81, whereas the stereotype «Type» is predefined in the UML 2 specification as part of the UML StandardProfileL2. The definitions in the standard ISO 19136 Annex E extend these stereotypes by several tag definitions. Using the UML package merge concept, only these additional tag definitions have to be defined in the proposed formal ISO 19136 UML profile. The extension relationships between the stereotypes and the corresponding UML metaclasses do not have to be defined anew.

Figure 5.10 shows the UML package which results conceptually from merging the merged packages *ISO/TS19103* and *StandardProfileL2* with the receiving package *ISO19136* from figure 5.7, page 95. In this process, the elements *ISO/TS19103::CodeList*, *ISO/TS19103::Union*, *ISO/TS19103::Leaf* and *StandardProfileL2::Type* from the merged packages *ISO/TS19103* and *StandardProfileL2* are merged with the elements *ISO19136::CodeList*, *ISO19136::Union*, *ISO19136::Leaf* and *ISO19136::Type* from the receiving package *ISO19136* into equally named resulting elements which are marked in red in the UML diagram. All other elements from the proposed formal ISO 19136 UML profile are simply copied into the resulting package. The stereotypes from the UML StandardProfileL2 are, apart from the stereotype «Type», not displayed in the UML diagram. Theoretically they are also copied into the resulting package, but are omitted to keep the UML diagram focused on the relevant aspects.

5.3.2.4 Formal definition of stereotypes for *no stereotype* indications

The standard ISO 19136 Annex E defines tag definitions to be applied to UML packages contained within UML application schemas (cf. section 5.3.1.1, page 87), UML classes representing object types (cf. section 5.3.1.3, page 89) as well as UML attributes and UML association ends (cf. section 5.3.1.7, page 92). However, no stereotypes are defined in ISO 19136 Annex E for these tag definitions. Since in the context of UML 2 tag definitions can only be applied to UML model elements in connection with a stereotype, the following stereotypes are introduced in the proposed formal ISO 19136 UML profile:

- The stereotype «Package» as extension of the UML metaclass *Package* to mark UML packages contained within UML application schemas,
- the stereotype «ObjectType» as extension of the UML metaclass *Class* to mark UML classes representing object types and
- the stereotype «Property» as extension of the UML metaclass *Property* to mark UML attributes and UML association ends.

The stereotype «Package» can be applied to UML packages without stereotype but also to UML packages which already have an arbitrary stereotype.

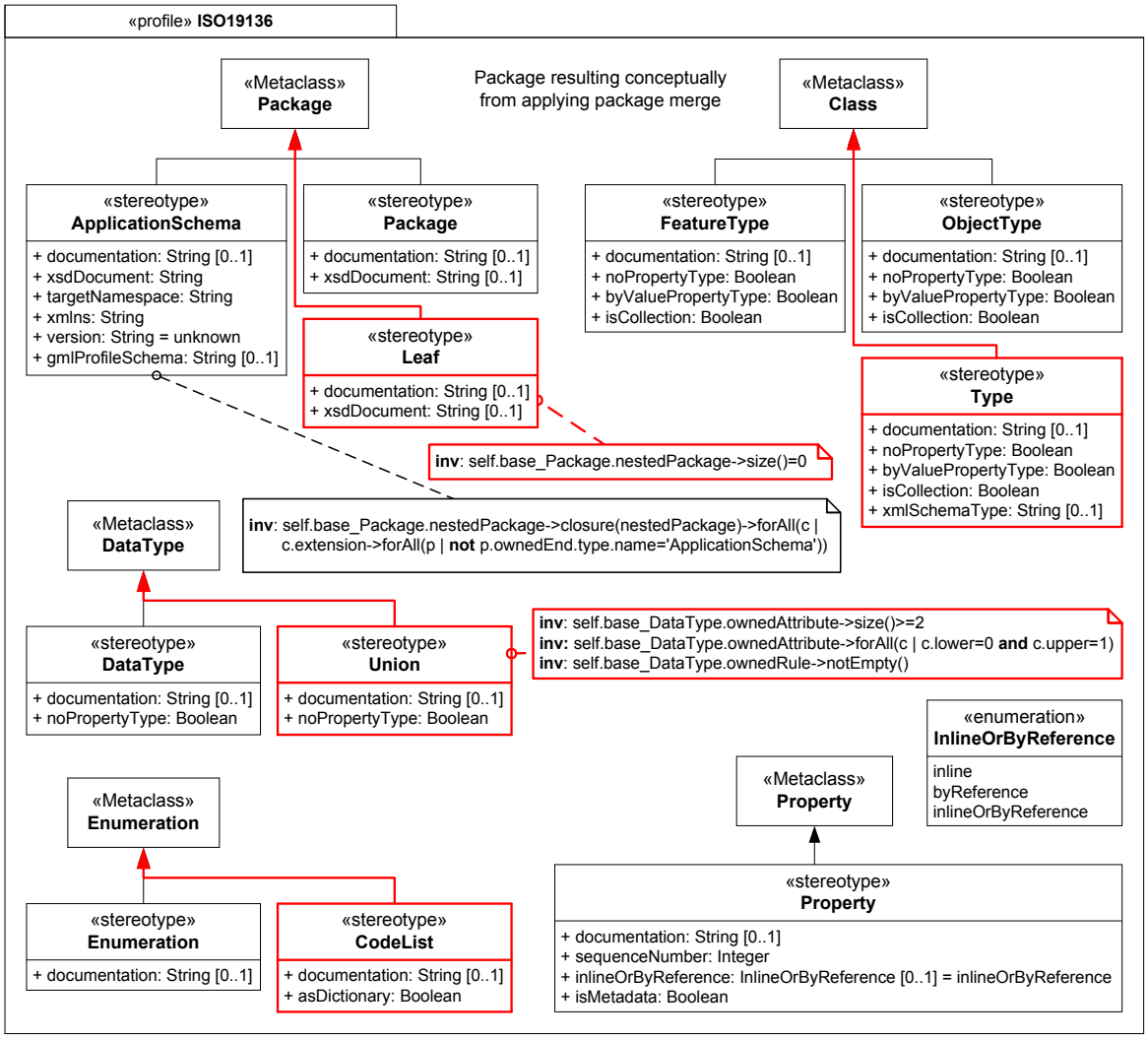


Figure 5.10: Proposed formal ISO 19136 UML profile, represented as UML package resulting conceptually from merging the merged packages *ISO/TS19103* and *StandardProfileL2* with the receiving package *ISO19136*

5.4 The INSPIRE UML profile

The European SDI initiative INSPIRE (European Parliament and Council 2007) provides its own UML profile which is based on the standard ISO 19136 Annex E. The INSPIRE UML profile is discussed here, as it is of importance to each EU Member State advised to provide its data according to the INSPIRE data specifications. Furthermore, INSPIRE recommends to use the 'INSPIRE Data Specifications as a basis for national or community extensions' (JRC 2014a), and it is also expected that the *INSPIRE Generic Conceptual Model* will influence 'modelling activities for spatial data at the national level, because it adds value to the national spatial data infrastructure and simplifies transformation to the INSPIRE Data Specifications' (JRC 2014a). Above that, also the use case in this thesis will also make use of the INSPIRE data specifications (cf. section 7.1, page 139).

The INSPIRE Generic Conceptual Model (GCM) is specified in the INSPIRE document *D2.5: Generic Conceptual Model, Version 3.4* (JRC 2014a). The GCM is a framework based on which the INSPIRE data specifications for each individual INSPIRE theme are to be defined. The framework, in turn, is based on the ISO 191xx series of geographic information standards. The GCM requires that every INSPIRE data specification provides at least one INSPIRE application schema defined using UML. The INSPIRE application schemas have to conform to the standards ISO 19109 and ISO/TS 19103, i. e. they have to be defined compliant to the General Feature Model of the standard ISO 19109 and they have to use the elementary data types of the standard ISO/TS 19103. However, the GCM also requires that the INSPIRE application schemas are defined using UML version 2.1.2, instead of UML version 1.4.2 (ISO/IEC 19501) on which the standards ISO 19109 and ISO/TS 19103 are based. Furthermore, the use of UML has to conform to the standard ISO 19136 Annex E.

The INSPIRE document *D2.7: Guidelines for the encoding of spatial data, Version 3.3* (JRC 2014b) specifies the default encoding rule for the INSPIRE application schemas. This default encoding rule corresponds to the encoding rule defined in the standard ISO 19136 Annex E and, in addition, comprises extensions to the ISO 19136 Annex E encoding rule defined in the GML version 3.3 specification (Portele 2012b) as well as additional rules defined in the document D2.7 itself. However, it is not obligatory for INSPIRE application schemas to make use of this encoding rule as the mandatory encoding rule. Depending on the use case of each specific INSPIRE application schema, also a different encoding rule could be specified as the mandatory encoding rule.

The INSPIRE UML profile to be used for modelling the INSPIRE application schemas is provided in the document D2.5 in tabular form, containing all relevant stereotypes and tag definitions. Table 5.11 is based on this overview, it lists all stereotypes which are part of the INSPIRE UML profile. Furthermore, a UML profile diagram was created for this thesis based on the information provided in the documents D2.5 and D2.7 which is displayed in figure C.6, page 188.

Although based on the standard ISO 19136 Annex E, the stereotype «Association» defined in ISO 19136 Annex E is not part of the INSPIRE UML profile provided in the document D2.5 and, thus, also not included in this table. Furthermore, it can be noticed, that some of the stereotypes refer to the UML 1 metaclasses *Dependency*, *Attribute* and *AssociationRole*, although the GCM requires the use of UML 2. Also, the stereotype names start with lower-case letter, in contrast to the stereotype names in the standards ISO/TS 19103 and ISO 19136 Annex E. Above that, the GCM recommends to complement packages and classifiers by the tag definition *xsdEncodingRule* to be able to indicate the encoding rule to be applied in the encoding of an INSPIRE application schema. Since the tag definition is just a recommendation and not a requirement, it is defined in the UML profile diagram with multiplicity [0..1].

In the following, the stereotypes defined new in the INSPIRE UML profile will be discussed in more detail. The stereotypes which were already defined in the standard ISO 19136 Annex E are basically only complemented by additional tag definitions in the INSPIRE UML profile; also, the

Table 5.11: Stereotypes of the INSPIRE UML profile

Stereotype	UML metaclass	Origin of stereotype
«applicationSchema»	Package	ISO 19136 Annex E
any or no stereotype	Package	ISO 19136 Annex E
«leaf»	Package	ISO/TS 19103
«import»	Dependency	UML 1
«featureType»	Class	ISO 19136 Annex E
«placeholder»	Class	INSPIRE D2.5
no stereotype	Class	ISO 19136 Annex E
«type»	Class	UML 1
«enumeration»	Class	UML 1
«codeList»	Class	ISO/TS 19103
«union»	Class	ISO/TS 19103 / ISO 19107
«dataType»	Class	UML 1
no stereotype	Attribute, AssociationRole	ISO 19136 Annex E
no stereotype	Attribute with value type «codeList»	INSPIRE D2.5
«voidable»	Attribute, AssociationRole	INSPIRE D2.5
«lifeCycleInfo»	Attribute, AssociationRole	INSPIRE D2.5
«version»	AssociationRole	INSPIRE D2.5

conclusions drawn in section 5.3.1.8, page 94, are still valid for these stereotypes. Therefore, they will not be discussed again. For completeness, however, the extensions defined in the INSPIRE UML profile for these stereotypes are listed in appendix C.3, page 186. After this discussion, a formal UML profile compliant with the UML profile definition of the OMG will be proposed.

5.4.1 Discussion of the INSPIRE-specific stereotypes

The INSPIRE UML profile defines several new stereotypes to be applied in modelling INSPIRE application schemas which will be presented in this section.

5.4.1.1 The stereotype «placeholder»

The INSPIRE document D2.5 defines the use of the stereotype «placeholder» as follows:

> All spatial object types specified in INSPIRE application schemas shall carry the stereotype <<featureType>>. In cases where a spatial object type acts as a placeholder for a spatial object type that will be specified as part of a future spatial data theme the stereotype <<placeholder>> shall be used. (JRC 2014a)

Table 5.12 lists the tag definitions defined in the INSPIRE document D2.5 for the stereotype «placeholder». Based on the above definition, the stereotype «placeholder» represents feature types which are to be defined in a future data specification. Thus, just as the stereotype «featureType», also the stereotype «placeholder» extends the UML metaclass *Class*. The stereotype «placeholder» as well as its related tag definitions can be defined formally as part of a UML profile compliant with the UML profile definition of the OMG.

Table 5.12: INSPIRE UML profile: Tag definitions of the stereotype «placeholder»

Name	Meaning	Type	Multiplicity	Use
noPropertyType	An XML complex type is to be generated carrying the name of the placeholder with the suffix PropertyType	Boolean	1	GML encoding
byValuePropertyType	An XML complex type is to be generated carrying the name of the placeholder with the suffix PropertyByValueType	Boolean	1	GML encoding
isCollection	The placeholder is to be encoded as GML feature collection	Boolean	1	GML encoding
inspireConcept	URI reference to the feature concept	String	1	Conceptual level, GML encoding
xsdEncodingRule	The encoding rule to be applied	String	0..1	GML encoding

5.4.1.2 The stereotype «voidable»

For the stereotype «voidable» the following definition is given in the INSPIRE document D2.5:

> If a characteristic of a spatial object may not be present in the spatial data set independent of its presence or applicability in the real world, the property shall receive the stereotype «voidable». If and only if a property receives the stereotype «voidable», the value of void may be used as a value of the property which shall imply that the characteristic is not present in the spatial data set, but may be present or applicable in the real world. It shall be possible to qualify a value of void in the data with a reason using the VoidReasonValue type [...]. (JRC 2014a)

The stereotype «voidable» defines rules for UML properties which exist in the real world, but whose value might not be present in the data set. In this case, the UML property will be assigned the value *Unknown*, when the property value is not known to the data provider, *Unpopulated*, when the value is not part of the data set provided by the data provider, or *Withheld*, when the value may be known to the data provider, but is confidential (JRC 2014a). All three values are defined in the code list *VoidReasonValue*, depicted on the left side of figure 5.11.

In section 5.3.1.7, page 92, it was discussed that stereotypes which are applied to UML attributes and UML association ends extend in UML 1 the UML metaclasses *Attribute* and *AssociationEnd*, whereas in UML 2 they extend the UML metaclass *Property*. Although INSPIRE requires the use of UML 2.1.2, the majority of the INSPIRE UML profile as defined in the document D2.5 represents an extension of the ISO 19136 UML profile which is based on UML 1.4.2. Therefore, in the INSPIRE UML profile diagram in figure C.6, page 188, the stereotype «voidable» is, for the moment, defined as an extension of the UML metaclasses *Attribute* and *AssociationEnd*. No tag definitions are defined for the stereotype. It should be noted here, that in the current implementation of the INSPIRE UML profile for use with the software Enterprise Architect (cf. appendix C.1, page 179) the stereotype «voidable» extends the UML metaclasses *Attribute* and *AssociationRole*. The UML metaclass *AssociationRole*, however, does not seem suitable here, as was discussed in section 5.3.1.7, page 92.

The stereotype «voidable» can be defined formally according to the UML profile definition of the OMG. However, the stereotype causes a semantic modification of the UML specification when applied

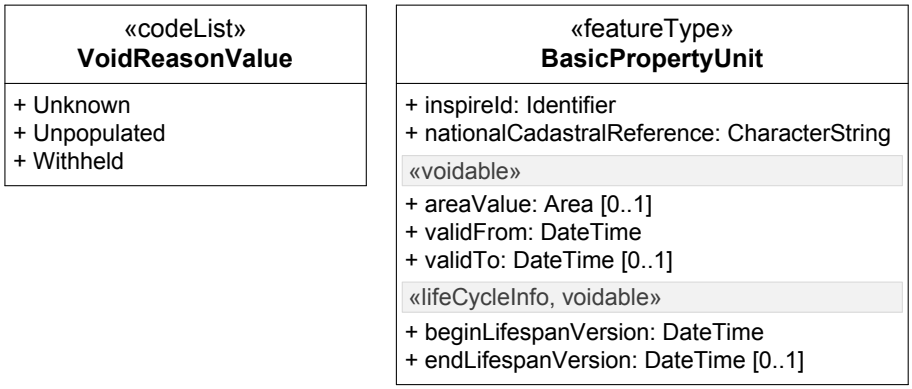


Figure 5.11: Stereotype «voidable» according to the INSPIRE UML profile (JRC 2014e) and its use with the feature type *BasicPropertyUnit* from the INSPIRE data specification Cadastral Parcels (JRC 2014d)

to a UML property, because the value range of the affected UML properties now not only consists of the actual type, but also of the values of the code list *VoidReasonValue*. Consider for example the property *beginLifespanVersion* of the feature type *BasicPropertyUnit* depicted on the right side of figure 5.11. The property is of the type *DateTime* and of the multiplicity 1 which means that the property exhibits *exactly one value* of the type *DateTime* in an instance. However, due to the fact that the stereotype «voidable» has been applied to the property, its value range not only comprises *DateTime* information any more, but also the values of the code list *VoidReasonValue* which contradict the type *DateTime*. The same holds true for all UML properties to which the stereotype «voidable» has been applied.

Although being technically justifiable, the semantics of the stereotype is not compliant with the UML specification and can affect the machine-interpretability of the INSPIRE application schemas (Kutzner and Donaubauer 2012). Furthermore, problems can occur during the transformation between different UML application schemas.

5.4.1.3 The stereotypes «lifeCycleInfo» and «version»

The last two stereotypes defined in the INSPIRE document D2.5 are «lifeCycleInfo» and «version». The document provides the following definitions for them:

> A property that is considered to be part of the life-cycle information of a spatial object shall receive the stereotype «lifeCycleInfo». (JRC 2014a)
>
> An association role that ends at a spatial object type shall imply that the value of the property is the spatial object unless the role has the stereotype «version» which shall imply that the value of the property is a specific version of the target spatial object. (JRC 2014a)

According to this definition, the stereotype «lifeCycleInfo» can be applied to UML attributes and UML association ends, whereas the stereotype «version» is only applied to UML association ends. Thus, as argued in the context of the stereotype «voidable», the stereotype «lifeCycleInfo» is defined as an

extension of the UML metaclasses *Attribute* and *AssociationEnd*, whereas the stereotype «version» only extends the UML metaclass *AssociationEnd* in the INSPIRE UML profile diagram in figure C.6, page 188. No tag definitions are defined for neither of the stereotypes.

Both stereotypes play a role in particular at the conceptual level, i. e. at the level of the INSPIRE application schemas, by labelling properties which provide life-cycle information and association ends which are connected to versioned feature types. The stereotypes are ignored by the default encoding rule and do not have any impact on the derivation of GML application schemas.

5.4.1.4 UML attributes of the value type «codeList»

The INSPIRE UML profile defines the tag definition *obligation* for UML attributes which are of the value type «codeList»[3]; however, without defining a stereotype and, thus, not compliant to the UML 2 profile definition of the OMG, as was already discussed in the context of the ISO 19136 UML profile (cf. sections 5.3.1.1, page 87, 5.3.1.3, page 89 and 5.3.1.7, page 92). The current situation is reflected in the INSPIRE UML profile diagram in figure C.6, page 188, by modelling the extension there without stereotype name as well.

5.4.1.5 Conclusion

The INSPIRE UML profile defined in the INSPIRE documents D2.5 and D2.7 reuses for the main part the UML profile which is defined by the standard ISO 19136 Annex E and which is extended by the GML version 3.3 specification. It complements the existing stereotypes by additional tag definitions required for encoding INSPIRE application schemas according to the default encoding rule proposed in the INSPIRE document D2.7. Therefore, regarding these existing stereotypes the conclusion drawn for the ISO 19136 UML profile (cf. section 5.3.1.8, page 94) also apply here. Furthermore, the issue of tag definitions for which no stereotype was defined applies here as well.

The INSPIRE UML profile defines four new stereotypes, each can be defined formally according to the UML profile definition of the OMG. The stereotypes «lifeCycleInfo» and «version» are relevant only for adding information to the INSPIRE application schemas at the conceptual level, whereas the stereotypes «placeholder» and «voidable» also influence the derivation of GML application schemas. However, the stereotype «voidable» causes a semantic modification of the UML specification when used in defining INSPIRE application schemas which affects their machine-interpretability and can cause problems during schema transformation. A solution needs to be found which prevents the semantic modification, but which at the same time preserves the semantics of the stereotype «voidable».

5.4.2 Proposed formal definition of an INSPIRE UML profile

In section 5.3.2, page 94, it was argued that package merge can be regarded as a convenient concept for creating UML profiles targeted at different communities within a certain domain – INSPIRE can in this context be regarded as a specific community within the geospatial domain. The INSPIRE

[3] Actually, the INSPIRE document D2.5 is not fully clear regarding the necessity of the tag definition *obligation*. On the one hand, the document lists the tag definition in the INSPIRE UML profile overview, but, on the other hand, it mentions in a note that the tag definition is no longer required due to the introduction of the tag definition *extensibility* as part of the stereotype «codeList». For this reason, the tag definition will only be considered in the discussion of the INSPIRE UML profile as defined currently, but will not be further considered in the formal INSPIRE UML profile proposed hereafter in section 5.4.2, page 104.

UML profile makes use of the ISO 19136 UML profile and of extensions to the ISO 19136 UML profile which are defined in a GML 3.3 UML profile. In addition, it extends the existing stereotypes by further tag definitions and defines additional stereotypes required specifically for being able to automatically derive GML application schemas from INSPIRE application schemas according to the default encoding rule proposed in the INSPIRE document D2.7.

Thus, in terms of the UML package merge concept, first the GML 3.3 UML profile (receiving package) needs to be merged with the ISO 19136 UML profile (merged package); afterwards the GML 3.3 UML profile (merged package) can be merged with the INSPIRE UML profile (receiving package) as is displayed in figure 5.12, resulting in a UML profile package which provides all necessary concepts required for being able to fully apply the INSPIRE default encoding rule.

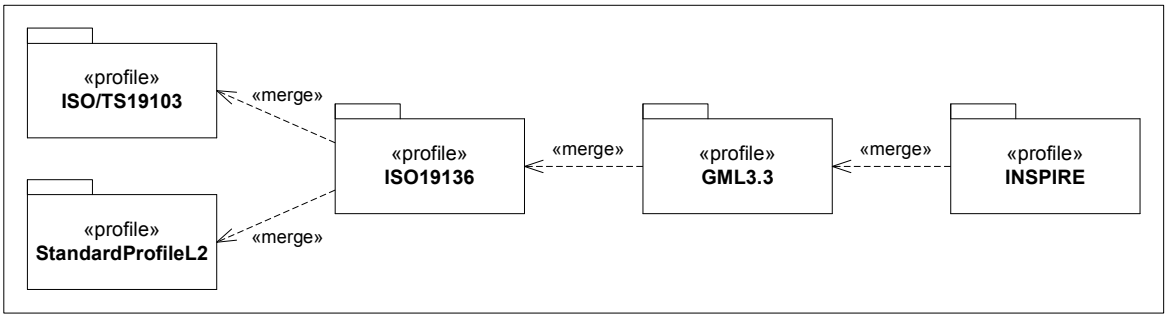


Figure 5.12: Merging of the ISO 19136 UML profile and the GML 3.3 extensions into the proposed formal INSPIRE UML profile

The formal INSPIRE UML profile proposed in this section takes into account the UML package merge concept. Figure C.7, page 189, displays the proposed formal INSPIRE UML profile. As suggested by the UML profile definition of the OMG, all stereotype names are started with upper-case letters in the proposed formal INSPIRE UML profile. Stereotypes which have already been defined in one of the merged packages are not defined anew, but only complement the existing definitions by the additional concepts defined in the INSPIRE documents D2.5 and D2.7.

Section 5.3.1.2, page 88, pointed out that «import» is not a stereotype, but a keyword. Since no tag definitions are defined in the INSPIRE documents D2.5 and D2.7, no necessity exists to define a specific stereotype providing tag definitions for this keyword; therefore, «import» has not been included in the proposed formal INSPIRE UML profile.

The INSPIRE document D2.7 specifies that when no value is provided for the tag definition *xsdEncodingRule*, the value *iso19136_2007_INSPIRE_Extensions* is the default. This is indicated in the proposed formal INSPIRE UML profile by setting the enumeration literal *iso19136_2007_INSPIRE_Extensions* from the enumeration *EncodingRule* as initial value of every stereotype which contains the tag definition *xsdEncodingRule*. Alternatively, an OCL constraint would have to be defined for every stereotype which specifies this condition. Listing 5.6 shows such an OCL constraint exemplarily for the stereotype «FeatureType».

Listing 5.6: OCL constraint stating the initial value of the tag definition *xsdEncodingRule* of the proposed formally defined stereotype «FeatureType»

```
context FeatureType::xsdEncodingRule : EncodingRule
    init: EncodingRule::iso19136_2007_INSPIRE_Extensions
```

The GCM requires that the values of the tag definitions *noPropertyType* and *byValuePropertyType* are always set to *false*. This affects the stereotypes «FeatureType», «Placeholder», «Type» and «ObjectType». Thus, in the proposed formal INSPIRE UML profile a suitable OCL constraint is defined for each stereotype which states this requirement, provided that the INSPIRE encoding rule (enumeration literal *iso19136_2007_INSPIRE_Extensions*) is chosen as encoding rule to apply on an INSPIRE application schema. Listing 5.7 shows the OCL constraint which was defined for the stereotype «FeatureType». In the same way, the tag definition *noPropertyType* of the stereotypes «DataType» and «Union» is set to *false*.

Listing 5.7: OCL constraint stating the tagged values of the tag definitions *noPropertyType* and *byValuePropertyType* of the proposed formally defined stereotype «FeatureType»

```
context FeatureType inv:
    self.xsdEncodingRule = EncodingRule::iso19136_2007_INSPIRE_Extensions implies
    self.noPropertyType = Boolean::false and self.byValuePropertyType = Boolean::false
```

The INSPIRE document D2.7 requires the tag definition *asDictionary* of the stereotype «codeList» to be set to true. Furthermore, when the tag definition *extensibility* has the value *any*, then the value of the tag definition *vocabulary* must be empty or the tag definition must be missing according to the GCM; in all other cases a value must be provided. This is expressed in the proposed formal INSPIRE UML profile by the OCL constraint displayed in listing 5.8.

Listing 5.8: OCL constraint stating the tagged values of the tag definitions *asDictionary*, *extensibility* and *vocabulary* of the proposed formally defined stereotype «codeList»

```
context CodeList
    inv: self.asDictionary = Boolean::true
    inv: if self.extensibility = Extensibility::any
        then self.vocabulary.size() = 0 or self.vocabulary.oclIsUndefined()
        else self.vocabulary.size() > 0
        endif
```

Furthermore, the INSPIRE document D2.7 recommends to assign the value *byReference* to the tag definition *inlineOrByReference* when the feature association end is navigable. This affects the stereotype «Property» for which the OCL constraint in listing 5.9 is defined. The OCL constraint makes use of the operation *isNavigable()* which is predefined in the UML specification.

Listing 5.9: OCL constraint stating the tagged value of the tag definition *inlineOrByReference* of the proposed formally defined stereotype «Property»

```
context Property inv:
    self.base_Property.isNavigable() implies inlineOrByReference = InlineOrByReference::byReference
```

As discussed in section 5.4.1.2, page 102, the stereotype «voidable» can be defined formally according to the UML profile definition of the OMG. However, the stereotype causes a semantic modification of the UML specification when applied to a UML property. To resolve the semantic modification, it is proposed here to introduce alternative types in the form of UML data types, one alternative type for every actual type. These alternative types consist of two UML properties, one

property yielding the actual type and one property yielding the *VoidReasonValue* type. The actual type of each property with the stereotype «voidable» is then replaced by the alternative type. Furthermore, these alternative types are provided with the stereotype «Union» as defined in section 5.1.2.2, page 83. In this way, the «voidable» properties can at instance level be assigned either of the two properties of the intermediate type, allowing to preserve the semantics of the stereotype «voidable» and to be compliant with the UML specification at the same time.

This is illustrated by using the example from figure 5.11, page 103, again. The value of the property *beginLifespanVersion* is either a *DateTime* value or a value from the code list *VoidReasonValue*. Therefore, the alternative type *VoidableDateTime* is introduced in figure 5.13. It contains two UML properties, the property *datetime* which yields the actual type *DateTime* and the property *voidReason* which yields the *VoidReasonValue* type. Furthermore, the actual type of the property *beginLifespanVersion* is replaced by the alternative type *VoidableDateTime*. By applying the stereotype «Union» as defined in section 5.1.2.2, page 83, to the alternative type *VoidableDateTime* and defining an appropriate OCL constraint, the property *beginLifespanVersion* can at instance level either be assigned the property *datetime* of the type *DateTime* or the property *voidReason* of the type *VoidReasonValue*. This approach means, that the semantics of «voidable» properties is not expressed completely any more within the feature type class that contains the «voidable» properties. To obtain a formally correct UML diagram, also the new «voidable» data types have to be included.

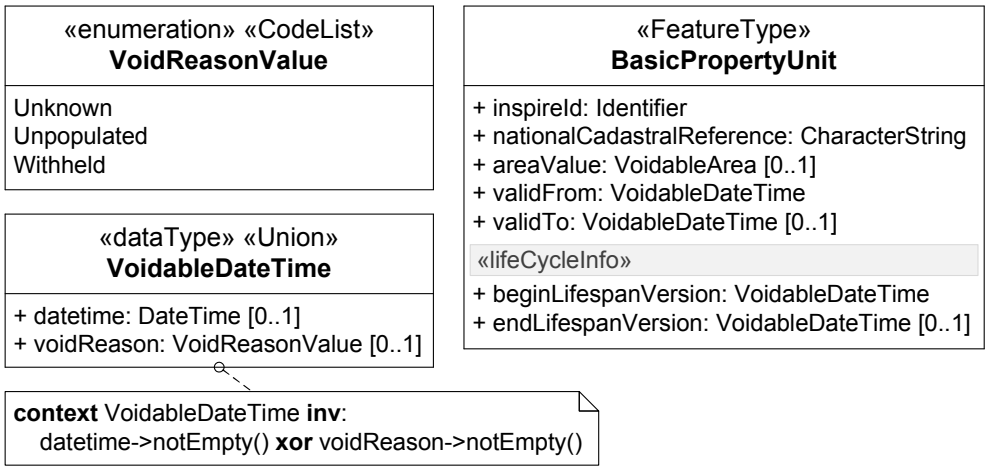


Figure 5.13: Code list *VoidReasonValue* and feature type *BasicPropertyUnit* from the INSPIRE data specification Cadastral Parcels (JRC 2014d) as well as the intermediate type *VoidableDateTime* based on the proposed formal definition of the stereotype «voidable»

5.5 Further relevant UML profiles used in SDIs today

Besides the UML profiles presented above in detail, several other UML profiles are in use in the geospatial domain which partially extend the above UML profiles by certain aspects only. This section shortly presents four further relevant UML profiles, the AAA UML profile, the INTERLIS UML profile, the ELF UML profile and the CityGML UML profile, highlighting and discussing their most important aspects.

5.5.1 The AAA UML profile

The German AAA reference model defines the geographic reference data to be provided by the federal mapping agencies of Germany. AAA is an acronym for *AFIS-ALKIS-ATKIS*; *AFIS* stands for the Official Geodetic Control Stations Information System of Germany, *ALKIS* for the Official Real Estate Cadastre Information System and *ATKIS* for the Official Topographic Cartographic Information System. These three information systems are associated in the AAA reference model. The AAA reference model is described in the document *Documentation on the Modelling of Geoinformation of Official Surveying and Mapping (GeoInfoDoc)* (AdV 2014). In December 2014, version 7.0.1 of the GeoInfoDoc was published, but not yet officially adopted as the new reference version of the AAA reference model; nevertheless, the descriptions provided in this thesis already base on GeoInfoDoc version 7.0.1 (in fact, the explanations given in GeoInfoDoc version 6.0.1 do not differ very much from the statements below).

The conceptual data model of the AAA reference model is specified by the *AAA application schema*. The AAA application schema consists of the AAA basic schema which provides basic concepts relevant for all other schemas, the AAA technical schema which defines the actual feature types, the AAA versioning schema and the NAS (Standards-based Data Exchange Interface) operations for data exchange and output. NAS is the transfer format for exchanging the geospatial data; its encoding is based on the standard ISO 19136 Annex E. The AAA application schema, furthermore, is based on the standards ISO/TS 19103 and ISO 19109, which means, it uses the ISO/TS 19103 stereotypes and elementary data types and it is also compliant to the ISO 19109 GFM. The AAA application schema conforming to GeoInfoDoc version 6.0.1 was defined using UML 1.4.2 (AdV 2009). The new version of the AAA application schema reportedly uses UML 2.4.1, however, GeoInfoDoc 7.0.1 itself still states UML 1.4.2 as the version to be used (AdV 2014).

The AAA basic schema extends the metaclass GF_FeatureType of the GFM by the metaclass AA_ObjektOhneRaumbezug (object without spatial reference). This new metaclass is required for being able to define feature types which are not allowed to have a spatial reference. Furthermore, the GeoInfoDoc defines that each AAA feature type with a spatial reference is only allowed to exhibit one geometry at most. Feature types containing more than one geometry require that an individual feature type is created for each geometry the feature type contains. This represents a restriction of the GFM, according to which each feature type is allowed to contain several geometries (cf. section 2.3.2, page 14).

The GeoInfoDoc states that, in addition to the stereotypes from the standard ISO/TS 19103, the stereotype «FeatureType» and all tag definitions from the standard ISO 19136 are used in the AAA application schema as well. Furthermore, the stereotypes «Request» and «Response» are specified for identifying messages exchanged by NAS operations; however, it is not stated which UML metaclass they extend. Above that, the tag definitions xsdEncodingRule and reverseRoleNAS are defined, but also for them it is not stated, to which stereotypes they belong; only two hints are given. From the statement that '[f]or all classes the UML Tagged Value "xsdEncodingRule" is set: "iso19136_2007" [...]' (AdV 2009), one may conclude that the tag definition xsdEncodingRule is to be added to all stereotypes which extend the UML metaclass *Class*. Likewise, from the statement '[n]on-navigable association roles are set to [...], and the UML Tagged Value "reverseRoleNAS" is set to "true"' (AdV 2009), it may be concluded that the tag definition reverseRoleNAS is only required for stereotypes extending the UML metaclass *Property*. In addition, the Enterprise Architect project containing the AAA application schema (AdV 2015) makes use of the stereotypes «applicationSchema» and «schema». Furthermore, several other tag definitions appear there, such as *AAA:Kennung* (identifier) or

AAA:Modellart (model type); they are relevant for the so-called AAA Tool which allows for encoding as well as generating feature catalogues and profiles of the AAA application schema. These tag definitions are mentioned in the documentation of the AAA Tool, but are not really specified there (Portele et al. 2014).

The encoding of the AAA application schema in the transfer format NAS takes place in two steps, with an implementation schema specific to the platform GML (ISO 19136 Annex E) created as intermediate step (cf. section 2.3.3, page 16). This procedure is necessary, since the AAA application schema contains UML constructs which are not supported by the GML platform. The transformation into the implementation schema includes, amongst others, that multiple inheritance is resolved, non-navigable association roles are made navigable, certain classes and attributes are eliminated and that some attributes receive a new type (AdV 2014).

Above explanations on the stereotypes and tag definitions used by the AAA application schema show that the German AAA reference model, in effect, defines a UML profile of its own which contains concepts specific to the AAA domain. This UML profile will be referred to as *AAA UML profile* in the following. However, the definitions available of the stereotypes and tag definitions are not very detailed and far-scattered and, thus, do not give a clear picture of how the AAA UML profile exactly looks. A more concise definition of the AAA UML profile and a UML profile diagram would, therefore, highly be desired. Furthermore, since the AAA UML profile is based on the ISO 19103 UML profile, which, as discussed in section 5.1.1, page 76, contains several deficits, they are passed on to the AAA UML profile as well.

5.5.2 The INTERLIS UML profile

INTERLIS was published in Switzerland in 1991 as a mechanism for exchanging data between land information systems. INTERLIS is a conceptual modelling language and a data transfer format for modelling and exchanging geospatial data. INTERLIS is a Swiss standard and exists in two versions, INTERLIS 1 (Swiss standard SN 612030) and INTERLIS 2 (Swiss standard SN 612031) (KOGIS 2006). The explanations in this thesis refer to INTERLIS 2.

In contrast to UML, which is a general-purpose modelling language, INTERLIS represents a DSL (cf. section 2.2.4, page 11). Furthermore, UML is a graphical language, whereas INTERLIS was specified as a textual language. Being a DSL, INTERLIS not only defines general concepts for creating data models, such as *Model*, *Topic*, *Class* and *Attribute*, but also provides a certain number of primitive data types, such as *TextType*, *NumericType* and *DateTimeType*, as well as geometry types, such as *CoordinateType*, *PolylineType* and *SurfaceType*. UML, in contrast, only defines the four primitive data types *Integer*, *String*, *UnlimitedNatural* and *Boolean* (cf. section 4.2.1.3, page 57). For this reason, models created using INTERLIS do not require the data types and geometry types defined in the standards ISO/TS 19103 and ISO 19107. Moreover, INTERLIS generally is not based on the ISO 191xx series of geographic information standards.

INTERLIS models can be encoded in the form of INTERLIS-specific transfer formats and also in the form of GML application schemas. When encoding INTERLIS models, no implementation schema needs to be created as intermediate step (cf. section 2.3.3, page 16); this is due to the fact that INTERLIS models are very precise (Kutzner and Eisenhut 2010), i. e. they do not contain any concepts which cannot be represented in the INTERLIS and GML platforms.

INTERLIS defines its own metamodel which is not based on MOF and does not represent an extension of the UML metamodel. The INTERLIS metamodel is defined using the INTERLIS language itself (KOGIS 2008). Besides, in the context of the software UML/INTERLIS editor

(KOGIS 2015) an extension of the UML 1.4.2 metamodel was defined, which allows, on the one hand, for creating INTERLIS models graphically using a UML visualisation and, on the other hand, for exporting them as textual INTERLIS models. This UML metamodel extension is not standardised, thus, only the textual INTERLIS models represent normative models, not their visualisations in the form of UML (Kutzner and Eisenhut 2010).

For this reason, also no standardised UML profile exists for INTERLIS. However, some of the concepts provided by INTERLIS can be transferred to corresponding stereotypes; in this way, a specific INTERLIS UML profile can be constructed for the INTERLIS language. The stereotypes of this INTERLIS UML profile were presented in (Kutzner and Eisenhut 2010). Table 5.13 lists those stereotypes together with the UML metaclasses they extend. In addition, according to (Kutzner and Eisenhut 2010), tag definitions are required to be able to define the characteristics of the INTERLIS-specific concepts more precisely; however, no tag definitions are provided there, which is why they are also not considered in the context of this thesis.

According to (Kutzner and Eisenhut 2010), the INTERLIS UML profile does not fully conform to the UML profile definition of the OMG since it contains concepts such as «GraphicDef» which modify the semantics of the UML specification. Therefore, the INTERLIS UML profile behaves equivalent to the ISO/TS 19103 UML profile which does not conform to the UML profile definition of the OMG due to the stereotypes «CodeList» and «Union».

Table 5.13: Stereotypes of the INTERLIS UML profile (Kutzner and Eisenhut 2010, modified)

Stereotype	UML metaclass	Meaning
«ModelDef»	Package	Definition of an application schema (may contain geospatial object types, but may also contain type definitions only (KOGIS 2006))
«TopicDef»	Package	Definition of a data basket containing ‘a specific part of reality’ (KOGIS 2006)
«MetaDataBasketDef»	Package	Definition of graphic symbols and of coordinate reference systems
«UnitDef»	Class	Definition of units of measure
«FunctionDef»	Class	Definition of functions for use in constraints, ViewDef or GraphicDef
«ViewDef»	Class	Definition of a view
«GraphicDef»	Class	Graphic description
«DrawingRule»	Property	Part of GraphicDef
«LineFormTypeDef»	Class	Definition of new forms of curve segments
«RunTimeParameterDef»	Class	Definition of data from the run-time system, e. g. of the current date for use in constraints

5.5.3 The ELF UML profile

The European Location Framework (ELF) project focuses on delivering harmonised geospatial data beyond the scope of INSPIRE from the European national mapping and cadastral agencies to the public and private sectors in Europe. Sectors considered within the ELF project by means of sample applications are Health Statistics, Emergency Mapping, Real Estate and Insurance (ELF Project 2015).

For this purpose, ELF defines application schemas which extend the INSPIRE application schemas by additional required information. The ELF application schemas are created using UML and are to follow the guidelines provided in the INSPIRE GCM document (cf. section 5.4, page 100).

In addition, ELF defines its own ELF UML profile which is specified in the document *ELF WP2 – Modelling guidelines* (Borrebæk 2014). According to this document, the ELF UML profile 'adds some additional tagged values to the INSPIRE UML profile' (Borrebæk 2014), which implies that the ELF UML profile is of the same extent as the INSPIRE UML profile. However, the 'full ELF UML profile' (Borrebæk 2014) listed in the document consists only of the stereotypes «applicationSchema», «featureType», «dataType», «codeList» and «association.end», whereas a screenshot in the same document showing the ELF UML profile imported in Enterprise Architect gives the impression that the ELF UML profile is of equal extent as the INSPIRE UML profile. Thus, also here, a more concise definition as well as a UML profile diagram would be very beneficial. The additional tag definitions are *suppress* for indicating that a specific feature type is to be suppressed from being encoded in the GML application schema and *profiles* for categorising classes, attributes and association roles into specific ELF level of detail profiles.

Similar to the INSPIRE UML profile and the AAA UML profile, also the ELF UML profile defines concepts which are specific to the ELF community only. Furthermore, since the ELF UML profile represents an extension of the INSPIRE UML profile, it contains all those deficits discussed in the context of the INSPIRE UML profile in section 5.4, page 100, as well.

5.5.4 The CityGML UML profile

The international standard *OGC City Geography Markup Language (CityGML)* (Gröger et al. 2012) specifies an application-independent information model for representing, storing and exchanging semantic 3D city and landscape models. The standard defines, on the one hand, a data model which is specified using UML and, on the other hand, an exchange format which in the CityGML version 2.0 is based on GML 3.1.1. The new CityGML version 3.0, which is currently under development, will be based on GML 3.2.1 (ISO 19136) and, thus, will make use of the encoding rule defined in the standard ISO 19136 Annex E for deriving GML application schemas from a conceptual UML application schema (Kutzner and Kolbe 2016; Löwner et al. 2014).

A successful application of the ISO 19136 Annex E encoding rule will require the CityGML UML model version 3.0 to be defined based on the ISO 19136 UML profile presented in section 5.3, page 86. In addition, the CityGML standard defines two new stereotypes, «ADEElement» and «ADE» (cf. section 4.2.1.1, page 54, for a description of the semantics of these stereotypes, where they were already discussed in the context of the semantic modification they cause). Thus, also CityGML specifies a UML profile of its own, namely the CityGML UML profile, which extends the ISO 19136 UML profile by concepts specific to the CityGML community only. This means, furthermore, that the CityGML UML profile not only inherits the deficits from the ISO 19136 UML profile discussed in section 5.3.1, page 86, but – by reversing the meaning of the UML generalisation concept – add its own semantic modification of the UML specification to the CityGML UML profile.

6 A universal approach to the application of UML profiles in modelling and information integration of geospatial data

The previous chapter identified problems which can lead to limitations in meta-interoperability between models to which differing UML profiles are applied. This chapter will present solutions which help to solve these problems. Beginning with a general discussion of the findings from the analysed UML profiles, a generic concept for developing UML profiles in a structured and reusable way will be presented afterwards, which also accepts the variety of UML profiles existing in the geospatial domain. Next, a new UML profile, called *Core UML profile*, will be introduced as a universally applicable, fundamental building block for geospatial data modelling and model-driven transformation of geospatial data. Finally, a *multi-level information integration* framework will be presented which allows for transforming between UML models based on different UML profiles.

6.1 On the use and abuse of UML profiles

The previous chapter showed that a wide variety of UML profiles is currently in use in the geospatial domain – more than covered in this thesis (an example of a further UML profile not discussed here is the NEN3610 UML profile which was defined for the Dutch base model for geoinformation (NEN3610) (Geonovum 2011)). However, the possibility of being able to define any imaginable UML profile does not only bring about advantages as regards their use, but can also result in disadvantages and deficits due to an abuse of the UML profile mechanism.

The discussion revealed several advantages of using UML profiles in the geospatial domain. One important advantage is that concepts which specifically exist in the geospatial domain, but which are not represented in the standard UML language, such as the feature type or the code list concept, can be represented in a UML model by means of corresponding stereotypes and can, furthermore, be complemented by additional information using tag definitions or be restricted using constraints. Thus, one can say that a UML profile makes the UML model to which it is applied, on the one hand, more descriptive to the user of the UML model, but, on the other hand, also more precise in the sense of more machine-interpretable as regards the derivation of data transfer formats or the transformation between UML models. Furthermore, the UML profile mechanism allows individual communities within the geospatial domain, such as the INSPIRE community or the CityGML community, to make use of these general geospatial domain concepts and, if required, to define additional concepts which are specific to their community only. These additional concepts are represented in the form of stereotypes as well.

These advantages, however, are faced by several disadvantages. The previous chapter identified several problems which can lead to limitations in meta-interoperability between models to which different UML profiles are applied. One important issue are stereotypes which modify the semantics of the UML metamodel such that the UML models do not conform to the UML specification any more and, as a result, are not fully machine-interpretable any more. This problem can be accounted

for by the way UML metaclasses are selected to be extended by stereotypes. (Selic 2007) states that UML metaclasses are often chosen because they simply match syntactically, with the result that the stereotypes can be misinterpreted by UML tools, a fact, which is also expressed in (Pardillo 2010) by the hypothesis that 'instead of selecting extension metaclasses by their semantic closeness, many profiles extend metaclasses due to a notational closeness to the target language' (Pardillo 2010). The discussion showed that this perception also applies to some of the stereotypes defined in the UML profiles prevalent in the geospatial domain, such as the stereotypes «CodeList» and «voidable» (cf. sections 5.1.1.1, page 77, and 5.4.1.2, page 102, respectively). Above that, differences regarding the UML metaclass to extend when UML versions 1 and 2 are involved can pose a problem, too. For instance, a UML attribute is represented in UML 1 by the UML metaclass *AssociationEnd*, whereas in UML 2 it is represented by the UML metaclass *Property* (cf. section 5.3.2.1, page 96).

Another deficit noticed during the analysis was that the various specification documents do not define the stereotypes formally as part of a UML profile and also do not provide a UML profile diagram. The documents rather specify the stereotypes together with possible tag definitions, constraints and UML metaclasses they extend in tabular form or even in the running text using natural language. This informal way of specifying UML profiles can lead to imprecise definitions and result in an ambiguous interpretation when trying to remodel UML profiles formally using a UML tool. An example is the tag definition *obligation* for which the INSPIRE document D2.5 version 3.4 contains contradictory information whether the tag definition is still part of the INSPIRE UML profile or not (cf. section 5.4.1.4, page 104). This problem corresponds to the risks identified regarding the documentation of encoding rules and transformation definitions, respectively (cf. section 4.2.2, page 60). However, the definition of formal UML profiles does not only require sufficient knowledge of the UML profile mechanism, but also of the UML metamodel itself. Otherwise, erroneous formal UML diagrams are the result as the example of the ISO/DIS 19103 UML profile diagram contained in the ISO/DIS 19103 revision document illustrates (cf. section 5.1.3, page 85).

The analysis also showed that, even if a specific UML profile makes use of the general geospatial domain concepts which are already defined as stereotypes within another UML profile, this specific UML profile needs to define the stereotypes anew to be able to use them in the development of UML models. These continual redefinitions together with a lack of knowledge regarding the UML profile mechanism can result in stereotypes which do not reflect the original stereotypes, but rather represent new stereotypes with a different abstract and concrete syntax, an example is the stereotype «Application Schema»/«ApplicationSchema» (cf. section 5.3.2.1, page 96), or even with a new semantics in case the stereotype is defined once with tag definitions and/or constraints and once without; this applies, for instance, to the stereotypes from the ISO/TS 19103 UML profile which are complemented by tag definitions in the ISO 19136 UML profile. Furthermore, when several UML profiles are to be applied to the same UML model, stereotypes which are defined in more than one of the applied UML profiles will also appear more than once in the selection list of stereotypes applicable to the corresponding UML model element since, according to the UML profile definition, they represent two distinct stereotypes, each of them defined within a different UML package and, thus, also within a different namespace.

Two approaches for how to cope with these advantages and disadvantages of using UML profiles in the geospatial domain are thinkable, (1) agreeing upon one common UML profile to be used by all communities within the geospatial domain, or (2) accepting the existence of a diversity of UML profiles and trying to find solutions for how to overcome the existing deficits. Both approaches will be discussed in more detail in the following.

6.1.1 Adoption of one common UML profile

One approach to solve the above mentioned issues is that all communities within the geospatial domain agree upon adopting one common UML profile which has to be used by each community. This common UML profile can be regarded as an all-embracing UML profile, i. e. a UML profile which combines all concepts from each individual community UML profile in the sense of a union set.

An advantage of this approach is that transformations would no longer need to take place in a cross-community environment since all communities use the same UML profile and, thus, belong to the same community and exhibit the same semantics. Furthermore, agreeing on one common UML profile would also require to agree on one common UML version to be used by all communities for the development of UML models. To avoid the currently existing semantic modifications of the UML metamodel in the common UML profile, attention would need to be paid from the beginning to the fact that the UML profile is created conforming to the UML profile definition of the OMG. This would require each concept from each community UML profile to be examined regarding its compliance to the UML specification and, in case a semantic modification occurs, to be redefined accordingly before it can be added to the common UML profile.

However, one should be aware of the fact that this common UML profile can possibly result in a very large UML profile, in particular, when the individual community UML profiles contain many differing concepts which all need to be included in the common UML profile. One drawback thereby is that this all-embracing UML profile might, in the worst-case, consist of a huge number of concepts, each of them required by one specific community only, compared to only a fraction of concepts equally relevant to all communities. A consequence could be that this common UML profile, although agreed upon, will not gain enough acceptance by each community within the geospatial domain, due to all those concepts included in the UML profile which are not of interest to the individual community. A community could then start to develop a profile in the sense of defining a subset of those concepts from the common UML profile which are relevant to that community, on the one hand, to provide advise to the members of the community which concepts to use and, on the other hand, to enable interoperability between UML models within the community – at the same time limiting interoperability outside the community, in particular, when several communities start to define their own subsets – which inverts the original idea of the common UML profile again.

Another risk is that some of the concepts could be understood differently by different communities, in particular when the semantics of the concepts are not defined precisely enough. This could lead to the development of UML models which explicitly assign the same stereotypes and, thus, the same syntax, but implicitly convey a different semantics, likewise reducing the interoperability between different communities. A further aspect to consider is the evolution of UML models or, in general, of the topics dealt within a community, which could eventually require the introduction of a new concept to a community UML profile. Once the common UML profile is in use, the new concept would need to be introduced to this common UML profile, which, in turn, would require each community to adapt its UML models to the updated common UML profile, even if the new concept is not relevant to any other community except the one which introduced the concept.

6.1.2 Acceptance of a variety of UML profiles

The second approach, in contrast, focuses on accepting the variety of UML profiles existing within the geospatial domain. This approach sets out from the assumption that the presence of the different UML profiles indicates that UML profiles geared at specific communities or areas of application are

simply required since not all concepts existing within a certain community can be squeezed into one certain UML profile. Thus, one can also say, the current variety of UML profiles proves their necessity. Furthermore, the variety of UML profiles in use could also be a sign of an absence of certain concepts within the modelling language UML, indicating an insufficient semantic expressiveness due to the semantic modification of the UML specification caused by some of the evaluated UML profiles. An example is the CityGML ADE concept together with the corresponding stereotypes provided as part of the CityGML UML profile (cf. sections 4.2.1.1, page 54, and 5.5.4, page 111). The superclass strategy used by the ADE concept to enrich existing CityGML feature types with additional properties can only be represented in UML models by means of semantically modifying the UML metamodel.

One possibility, in particular to solve the problem of semantic modification of the UML metamodel, is the use of flexibly specifiable modelling languages such as DSLs which allow for defining community-specific metamodels. These metamodels can be designed such that they are able to express exactly the semantics required for defining geospatial data models within a specific community, preventing the need for an inappropriate extension of the UML metamodel. To enable transformations between models defined using different DSLs, they need to be based on the same meta-metamodel. However, this alone will not solve all limitations regarding meta-interoperability. Given that one community includes concepts in its DSL which are not included in the DSL of another community, a complete transformation can still not be guaranteed as the concepts of the first DSL might not adequately be transformable to the concepts of the second DSL. Furthermore, the development of a specific DSL also requires the development of specific tools, such as editors or transformation/encoding tools, to be able to make use of the DSL at all. Examples are the DSLs INTERLIS (cf. section 5.5.2, page 109) and HML (cf. section 4.3.4, page 70) for which specific tools such as editors needed to be developed.

Since a migration from UML and UML profiles to community-specific DSLs would involve immense time and effort, in particular in SDI initiatives which have advanced far already, a solution will be proposed in the remainder of this chapter, which explicitly supports a continued use of the various community-specific UML profiles in modelling as well as in information integration of geospatial data. This solution does not only help avoid the existence of different UML versions, but also helps reduce the amount of semantic modifications occurring in UML profiles and the negative effect they have on transformations. Furthermore, the use of UML instead of specific DSLs facilitates the use and information integration of geospatial data with external domains. One part of this solution focuses on designing the UML profiles in a more structured way. This structured definition is based on a set of core concepts which allows for reusing already defined concepts and for distinguishing between concepts relevant at the conceptual level and concepts relevant for the encoding only. The second, more comprehensive part of this solution deals with the development of a framework which allows for conducting cross-community transformations based on these core concepts and against the background of the existing variety of UML profiles.

6.2 Generic concept for the development of UML profiles

This section introduces a generic concept for developing UML profiles in a structured and reusable way which exhibits several advantages, compared to the way UML profiles are currently defined. The generic concept is deduced from a classification of those UML profiles which were analysed in the previous chapter. This classification will also be presented in the following.

6.2.1 Classification of UML profiles

The examination of the UML profiles in the previous chapter revealed several differences regarding the scope of the concepts defined by these UML profiles. While some concepts focus on the representation of general conceptual aspects in UML models, others serve in adding community-specific aspects to UML models and still others focus on the provision of encoding aspects. Based on these findings, the UML profiles can be classified into the following categories:

- *Base UML profiles*: The ISO/TS 19103 UML profile provides the base concepts *CodeList*, *Union* and *Leaf* which are reused by all other UML profiles. These base concepts are primarily relevant at the conceptual level for modelling geographic information using UML. Similarly, also the standard ISO 19109 provides two base concepts, *ApplicationSchema* and *FeatureType*, which are reused by all other UML profiles. These base concepts are relevant at the conceptual level for modelling geographic information in the form of UML application schemas compliant to the General Feature Model. However, these two base concepts are currently only defined in natural language, but not yet in the form of stereotypes as part of a formal UML profile. This will most likely be corrected in the revision of the standard ISO 19109 (cf. section 5.2, page 85). This classification and also the generic concept presented later in this section assume that such an ISO 19109 UML profile, which formally defines the concepts *ApplicationSchema* and *FeatureType* as stereotypes «ApplicationSchema» and «FeatureType», already exists. Since both, the ISO/TS 19103 UML profile and the ISO 19109 UML profile, provide base concepts to which all other UML profiles refer, one can also say that these two UML profiles represent base UML profiles which form the fundament relevant for the entire area of geographic information modelling using UML. When comparing these two UML profiles with the modelling levels of MDA, it can be noticed that the base concepts add semantics to the UML models solely at the platform-independent level, thus, both UML profiles can be located at the PIM level of MDA.
- *Encoding UML profiles*: The ISO 19136 UML profile reuses the concepts from the base UML profiles and extends them by concepts which are mainly required for enabling an automatic derivation of GML application schemas from UML application schemas. These extensions do not add semantics to UML application schemas needed at the conceptual level, they only add encoding-specific information to UML application schemas. For this reason, one can also say that the ISO 19136 UML profile represents an encoding UML profile which turns a platform-independent UML application schema into a platform-specific UML application schema located at the PSM level of MDA. In the same way as the ISO 19136 UML profile defines an encoding rule for deriving GML application schemas, further encoding UML profiles can be defined which provide encoding rules for other formats commonly used in geographic information, such as KML, Shape or GeoJSON.
- *Community UML profiles*: The INSPIRE UML profile, in turn, reuses the concepts from the base UML profiles as well as from the ISO 19136 UML profile and extends them by INSPIRE-specific concepts. Since these extensions are solely required in the context of INSPIRE, one can also say that the INSPIRE UML profile represents a community UML profile which adds community-specific information to the UML application schema. However, a distinction has to be made between extensions aimed at the conceptual or platform-independent level (such as the stereotypes «lifeCycleInfo» and «version») and extensions intended for the encoding or platform-specific level (such as the tagged value *iso19136_2007_INSPIRE_Extensions*). To establish a proper basis for the generic concept presented below, a further classification of community UML profiles into *community conceptual UML profiles* and *community encoding UML profiles* is determined here. The community conceptual UML profiles are only allowed to contain those concepts which are

relevant at the conceptual level and, thus, are located at the PIM level, whereas the community encoding UML profiles solely contain encoding concepts and, thus, are placed at the PSM level. Other community UML profiles analysed in the previous chapter are the AAA UML profile which also needs to be split into a community conceptual and a community encoding UML profile, the CityGML UML profile which only adds conceptual concepts to the UML profiles it extends and, thus, is located at the PIM level as well as the ELF UML profile which, in contrast, only adds encoding concepts to the UML profiles it extends and, therefore, is placed at the PSM level. In the same way, further community UML profiles can be defined which provide extensions valid only to specific communities.

The classification shows that different UML profiles fulfil different tasks by adding base information, encoding-specific information or community-specific information to UML models. These findings are underpinned by a classification of stereotypes depending on their role and their expressiveness they play in software development provided in (Staron and Kuzniarz 2005). According to this classification each stereotype belongs to one of three roles, namely code generation (the stereotype provides information which is required by code generators), metamodel specialisation (the stereotype introduces concepts which are relevant to a certain domain or platform) or model simplification (the stereotype serves in labelling model elements to distinguish them from other model elements). In addition, each stereotype is further distinguished by its expressiveness into a decorative (the stereotype changes the concrete syntax of the extended metaclass), a descriptive (the stereotype specialises the abstract syntax of the extended metaclass) or a restrictive stereotype (the stereotype specialises the semantics of the extended metaclass).

Table 6.1 provides an overview of the classification introduced above. First of all, the UML profiles are classified into the MDA levels PIM and PSM. All UML profiles which add conceptual semantics to UML models are placed at the PIM level, where they are, in turn, subclassified into base UML profiles and community conceptual UML profiles. In the same way, all UML profiles which add encoding-specific information to UML models are placed at the PSM level and, in turn, are subclassified into general encoding UML profiles and community encoding UML profiles. All UML profiles examined in the previous chapter can be allocated to one of these categories without problems, as is shown in the table. In addition, three other UML profiles commonly used in the geospatial domain are included in this table, the ISO/TS 19139 UML profile, which provides an encoding rule for metadata according to the standard ISO 19115, the GML 3.3. UML profile, which extends the ISO 19136 UML profile by additional concepts, as well as the ISO 19118 UML profile, which provides an encoding rule for deriving XML Schema documents from UML models. These three UML profiles belong to the category of general encoding UML profiles.

6.2.2 Modular construction of UML profiles based on the UML package merge concept

One of the problems mentioned in section 6.1, page 113, is that even if a specific UML profile is based on another, already existing UML profile, this specific UML profile needs to define the stereotypes from the existing UML profile anew to be able to use them in the development of UML models. To overcome this problem, a generic concept will be presented in the following which allows for constructing UML profiles in such a modular way that a redefinition of already existing stereotypes will no longer be required and that also the drawbacks going along with the redefinition will no longer occur. The concept makes use of UML package merge (cf. section 2.4.3, page 23) which allows for combining the contents of two UML packages into one UML package. Since UML profiles are UML

Table 6.1: Classification of UML profiles

MDA level	Category	UML profile
PIM	Base UML profiles	ISO/TS 19103 UML profile ISO 19109 UML profile
	Community conceptual UML profiles	INSPIRE UML profile AAA UML profile INTERLIS UML profile CityGML UML profile
PSM	General encoding UML profiles	ISO 19136 UML profile ISO 19139 UML profile GML 3.3 UML profile ISO 19118 UML profile
	Community encoding UML profiles	INSPIRE UML profile AAA UML profile ELF UML profile

packages themselves, package merge can be applied to UML profiles without any problems. Also, the proposed formal UML profiles for ISO 19136 and INSPIRE (cf. sections 5.3.2, page 94, and 5.4.2, page 104, respectively) already applied package merge and proved that it is a convenient concept for creating UML profiles in a modular way.

Figure 6.1 displays the generic concept. The concept is based on the finding that all UML profiles in use in the geospatial domain can be classified into one of the four different categories introduced in table 6.1. At the top level, the *base UML profiles* are located, i. e. the *ISO/TS 19103 UML profile*, which contains the concepts «CodeList», «Union» and «Leaf», as well as the *ISO 19109 UML profile*, which specifies the concepts «ApplicationSchema» and «FeatureType». Currently, when the concepts of both UML profiles are to be used for developing UML application schemas, they are defined anew within one single UML profile which is then applied to the UML model. When using package merge, in contrast, this redefinition is not required any more. The stereotypes from the ISO/TS 19103 UML profile are merged with the stereotypes from the ISO 19109 UML profile, which means, the latter UML profile is simply extended by the stereotypes from the former UML profile, the ISO 19109 UML profile resulting from this merge containing all five stereotypes[1]. Now, only this resulting UML profile needs to be applied to the UML application schema. In this way, the concepts from both UML profiles can be used in the development of UML application schemas, but without having to define the stereotypes anew. Since the standard ISO 19109 bases itself on the standard ISO/TS 19103 and the concepts defined therein, it was decided that the ISO/TS 19103 UML profile represents the merged package, whereas the ISO 19109 UML profile represents the receiving package. This allows for applying the ISO/TS 19103 UML profile also to UML models which only need to conform to the standard ISO/TS 19103, but which do not necessarily have to represent ISO 19109 compliant UML application schemas.

[1]Please note that the merge is carried out implicitly only, i. e. the ISO 19109 UML profile is not modified in reality as was explained in the introduction to the UML package merge concept (cf. section 2.4.3, page 23).

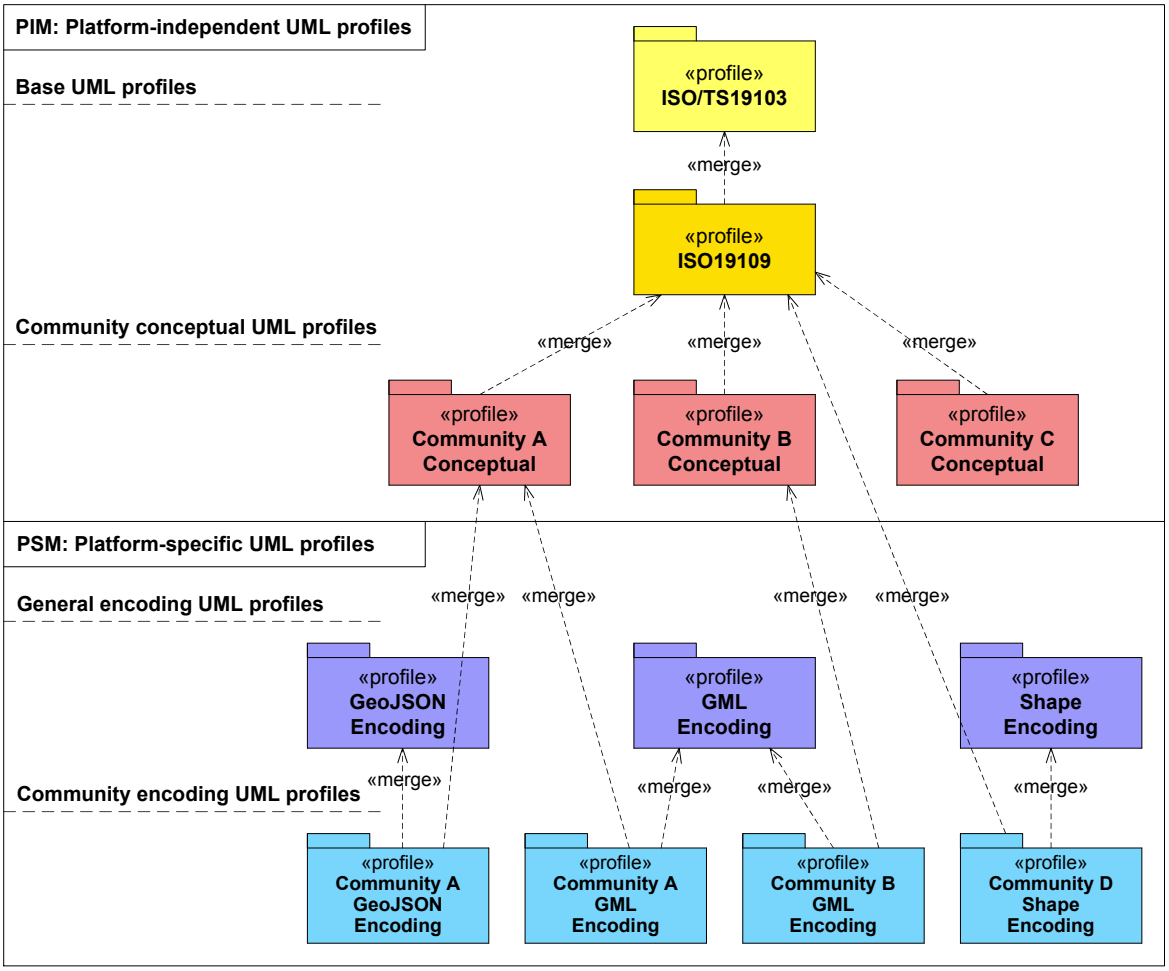


Figure 6.1: Generic concept for the modular construction of UML profiles using UML package merge

One layer below, the *community conceptual UML profiles* are located, i. e. those UML profiles which define community-specific concepts relevant at the conceptual level. Due to the use of UML package merge, the community conceptual UML profiles now only need to define those stereotypes which complement the ISO 19109 UML profile by additional information. To make use of a specific community UML profile when creating a UML application schema, the community UML profile is merged with the ISO 19109 UML profile and the UML profile resulting from the merge is then applied to the UML application schema. Regarding the German AAA community (cf. section 5.5.1, page 108), for instance, this means that a AAA conceptual UML profile needs to be created which merely specifies the AAA-specific stereotypes, the base stereotypes are included through merging the ISO 19109 UML profile with the AAA conceptual UML profile.

It is also thinkable, that a community exists which does not require the definition of community-specific concepts, but simply would like to make use of the base concepts alone. In this case, the community can apply the ISO 19109 UML profile directly to its UML application schemas. Another possibility, however, is that the community defines an empty community conceptual UML profile, i. e. a UML package without content, merges the ISO 19109 UML profile with this empty UML profile and applies the UML profile resulting from this merge – which, in fact, only exhibits the

stereotypes from the ISO 19109 UML profile – to its UML application schemas. This second option has the advantage that, although the community currently does not require community-specific concepts, the community can at least express that a community conceptual UML profile exists for the community. This is of advantage, in particular, when geospatial data are to be transformed between different communities (section 6.4, page 129). Furthermore, should it become necessary in the future to introduce community-specific concepts, the corresponding community conceptual UML profile already exists.

For creating conceptual UML application schemas at the PIM level, these community conceptual UML profiles are sufficient. However, when in a later step a data format schema is to be derived from a UML application schema, the encoding UML profiles located at the PSM level come into play. As defined in table 6.1, two categories exist, the *general encoding UML profiles* which add encoding-specific information aimed at specific data formats to a UML application schema, and *community encoding UML profiles* which extend the general encoding concepts by community-specific encoding requirements. To be able to derive now a specific data format schema from a UML application schema, the community conceptual UML profile needs to be merged with the relevant general encoding UML profile into a community encoding UML profile, converting the platform-independent UML application schema into a platform-specific UML application schema in this way[2]. This platform-specific UML application schema corresponds to the concept of the implementation schema (cf. section 2.3.3, page 16). Using the example of the AAA community again, this means that a general ISO 19136 UML profile is required which contains the general encoding-specific information for the GML format and a AAA GML encoding UML profile which specifies the AAA-specific encoding requirements for the GML format. The AAA encoding UML profile can then be merged with the AAA conceptual UML profile and the ISO 19136 UML profile, the UML profile resulting from this merge combining the content of these three UML profiles.

The general encoding UML profiles exist only once per data format. This is illustrated in figure 6.1 which contains example UML profiles for encoding UML application schemas in the formats GeoJSON, GML (equivalent to the ISO 19136 UML profile) and Shape. In this way, the general encoding UML profiles can be reused by every community which wishes to provide its geospatial data in one or several of these data formats. In the figure, the GML encoding UML profile is, for instance, used by community A as well as by community B. Community encoding UML profiles, in contrast, exist as many as data formats are to be derived from the UML application schemas within a specific community as they contain community-specific encoding aspects. In the figure, community A intends to provide its geospatial data, for instance, in the formats GeoGSON and GML; therefore, community A needs to define one community encoding UML profile for GeoJSON and another one for GML. Community B, although providing its geospatial data in the GML format as well, needs to specify its own community GML encoding UML profile as it may contain requirements which differ from those of community A. Furthermore, just as community conceptual UML profiles can be empty, also community encoding UML profiles can be without content, which is the case when a community does not have any community-specific encoding requirements. In this case, the community encoding UML profile resulting from the merge contains simply the content from the community conceptual UML profile and from the general encoding UML profile. Another example in the figure

[2]It can be assumed that the transformation of a PIM model into a PSM model will require more than only the application of platform-specific stereotypes and tag definitions. An example is the elimination of multiple inheritance when a PIM model is transformed into a PSM model targeted at the platform GML, which is applied to the AAA reference model in Germany (cf. section 2.3.3). These transformation steps will, however, not be considered in the context of this generic concept for the modular construction of UML profiles.

is demonstrated by community D. This community did not define a community conceptual UML profile, but merges directly the ISO 19109 UML profile and the general encoding UML profile with the community encoding UML profile.

6.2.3 Advantages of a modular UML profile construction

One advantage of this generic concept for the definition of UML profiles is the clear separation between conceptual information and encoding information. In the first stage of the modelling process, UML models can be defined fully platform-independent by solely applying a community conceptual UML profile. In the second stage of the modelling process, a platform-specific UML model can then be created by merging the community conceptual UML profile with a specific general encoding UML profile and a specific community encoding UML profile. The concept is fully compatible with the general MDA approach presented in section 3.1.2, page 29, and also with the example MDA transformation workflow displayed in figure 3.3, page 31, where a PIM model describing cadastral information is transformed into PSM models for the platforms GML, GeoJSON, Java and Relational, from which afterwards the corresponding formats and programming code are derived. To the PIM model, a community conceptual UML profile is applied, whereas to each of the PSM models a different community encoding UML profile is applied.

The clear separation between conceptual and encoding information also facilitates the process of information integration of geospatial data. A PIM model which includes both kinds of information may hinder the creation of transformation definitions, as it might be difficult to differentiate between conceptual information relevant for the transformation and which, thus, has to be considered in the creation of the transformation definitions, and encoding information which can be ignored since it is only important for deriving a data format schema from the source or destination UML model.

Furthermore, a possible evolution of UML profiles can benefit from a separation between community-specific conceptual/encoding information and general conceptual/encoding information as well. Evolution of a UML profile can occur, when new definitions are to be added to a community conceptual UML profile or when the community encoding is to be changed and, thus, the community encoding UML profile needs to be adapted. An example is the tag definition *extensibility* of the stereotype «codeList» which was introduced to the INSPIRE UML profile only in INSPIRE document D2.5, version 3.4 (cf. section 5.4.1.4, page 104). Other possibilities are that a new definition it to be integrated in the base UML profiles, that the general encoding rule for a specific format is to be changed or that a new version of a specific data format is published. In all these cases it is of advantage when only that particular UML profile needs to be modified which is primarily affected by the change. Imagine two different UML profiles which are created without using UML package merge. UML profile X contains definitions from community A and applies a GML encoding. UML profile Y contains the same definitions from community A, but applies a GeoJSON encoding. A change to certain community-specific definitions would now require a modification of the UML profiles X and Y to reflect the change. In contrast, when the generic concept introduced in this section is applied, only the conceptual UML profile of community A needs to be adapted and, depending on the UML tool used, the merge with the encoding UML profiles might need to be redone to propagate the change to the receiving packages.

When a new version of a specific data format is published, also the creation of a new general encoding UML profile is conceivable, instead of modifying the existing general encoding UML profile. The GML 3.2.1 encoding UML profile could, for instance, be complemented by an additional GML 3.3 encoding UML profile. In this case one community encoding UML profile would then

merge the GML 3.2.1 encoding UML profile and a second community encoding UML profile would merge the GML 3.3 encoding UML profile. In this way, the encoding information of both versions of the data format can be preserved, which is of advantage when geospatial data encoded in both versions are to be provided by that community.

Above that, depending on the encoding tool used, there may be the case that encoding information needs to be integrated in the encoding UML profile, without which the encoding tool is not able to execute the encoding correctly. This encoding information may only be relevant to one specific encoding tool, but not to another encoding tool, which, in turn, may require completely different encoding information. Here, particular *tool-specific UML profiles* are conceivable which only contain that part of information necessary to the particular encoding tool.

6.3 A universally applicable Core UML profile for geospatial data modelling

The previous section introduced a modular concept for the definition and use of UML profiles which accepts the variety of UML profiles existing in the geospatial domain. It, furthermore, became apparent that the ISO/TS 19103 UML profile and the ISO 19109 UML profile are always used jointly when UML application schemas compliant to the GFM are to be developed. For this reason, it actually would make sense to represent the concepts of these two UML profiles as one independent UML profile, which then constitutes a universally applicable, fundamental building block for geospatial data modelling and model-driven transformation of geospatial data. Since one can also say that these concepts are at the core of the geospatial domain, this UML profile will in the following be referred to as *Core UML profile*.

Such a Core UML profile offers several advantages. First of all, the Core UML profile is applicable to the definition of UML application schemas for communities which do not require community conceptual UML profiles. The Core UML profile can in this case be used as alternative to the ISO 19109 UML profile resulting from the proposed merge approach. Furthermore, individuals and, in particular, external domains which intend to model geospatial data, but do not have the resources to get familiar with the various concepts existing, or are not sure which concepts to use, can simply apply this Core UML profile and are, in this way, automatically provided with the most fundamental and relevant concepts. Not only can they feel confident that UML application schemas developed based on this Core UML profile will be understood by the geospatial domain, but also that the geospatial data which conform to these UML application schemas will be compatible with any other geospatial data conforming to UML application schemas developed based on the Core UML profile. Above that, there might be applications which require formal UML application schemas, but only encounter geospatial data or data specifications which are not yet formally defined. In this case, the Core UML profile can be applied in reverse-engineering corresponding UML application schemas from the available geospatial data or data specifications (cf. section 7.2.1, page 140, for an example). This method of using the Core UML profile can in a certain way be characterised as ad-hoc modelling of UML application schemas. Furthermore, the Core UML profile can be made use of when transforming between UML application schemas which are based on different UML profiles. Since the Core UML profile represents the least common denominator of the various UML profiles existing in the geospatial domain, it will guarantee a minimum of compatibility between them, and, thus, facilitate semantic transformations on the basis of a common base representation, as will be presented in section 6.4.

To reduce the limitations in meta-interoperability between models to which differing UML profiles are applied, two requirements need to be fulfilled by the Core UML profile. First, the Core UML profile needs to be defined as a formal UML profile conforming to the UML profile definition; second, the UML version based on which the UML profile will be developed needs to be fixed. The Core UML profile can be defined using package merge as described in the generic concept for the development of UML profiles (cf. section 6.4, page 129). However, this generic concept makes use of the profiles as they are currently defined. Since the Core UML profile might in principle contain further concepts which are fundamental at the conceptual level, it is advisable to start the development of the Core UML profile afresh.

Two methods of defining UML profiles are prevalent. The first method is to create the UML profile directly as described in section 2.4.1, page 19, whereas the second method consists of two steps; first, a domain model is defined which contains all those concepts required within a domain, afterwards, these concepts are mapped to appropriate concepts of the UML metamodel (Lagarde et al. 2008). In (Fuentes-Fernández and Vallecillo-Moreno 2004) useful guidelines are provided for how to define UML profiles using the second method. Similarly, (Selic 2007) describes how to define UML profiles systematically using the second method, avoiding in this way deficits, such as a low quality or technical errors, which often exist in UML profiles and which can prevent a UML profile from being properly processable with standard UML tools. Since the domain model represents a metamodel, (Selic 2007) suggests to define the domain model using MOF. After having defined the domain model, the UML profile is created by selecting for each concept in the domain model the most appropriate concept within the UML metamodel. During this process, it has to be taken care that the semantics of the chosen UML metaclass is as close as possible to the semantics of the domain concept and, also, that the attributes, associations and constraints of the chosen UML metaclass are not in conflict with the domain concept.

In the previous chapter, the first method was chosen for proposing formal UML profiles since the aim was not to create completely new UML profiles for the geospatial domain, but to analyse and enhance the existing UML profiles. Similarly, the intention of the Core UML profile to be developed in this section is not to define completely new and different concepts, but to define concepts which are derived from the UML profiles analysed in the previous chapter. Nevertheless, the second method will be applied in defining the Core UML profile, on the one hand, to demonstrate this two-step approach and, on the other hand, to allow for starting the development from the beginning, as was recommended above.

6.3.1 A domain model of relevant core concepts for the geospatial domain

In (Kutzner and Eisenhut 2010) it is suggested that the Core UML profile is only allowed to exhibit concepts which are common to all UML profiles and UML versions used in the geospatial domain and which means that the Core UML profile basically represents an intersection of all concepts specified within the existing UML profiles. This is seen as important since a comprehensive UML profile, which also contains concepts not shared by all UML profiles, could hinder a widespread use of geospatial data beyond the geospatial domain.

Possible disadvantages of such a comprehensive UML profile can, in part, be derived from the above presented advantages the Core UML profile yields. One disadvantage is that, depending on the number of concepts it contains, a thorough familiarisation with the comprehensive UML profile will be necessary by external domains prior to be able to make use of this UML profile. This, in turn, might require the preparation of a specification document which defines the syntax and semantics

of each individual concept of the UML profile in detail (similar to the UML profile specifications of the OMG which exist for several UML profiles and can comprise up to several hundred pages). Furthermore, a comprehensive UML profile can hinder the integration of geospatial data into external domains since, on the one hand, the concepts from the geospatial domain and from external domains may deviate such substantially that difficulties in matching them will arise and since, on the other hand, the creation of transformation definitions can result in a complex task, even if the concepts are reflected in external domains in one way or another. In addition, the disadvantages stated regarding the adoption of a common all-embracing UML profile (cf. section 6.1.1, page 115) apply here as well.

For these reasons, the domain model presented in the following will define a very manageable set of concepts only and, thus, will be very slender. Figure 6.2 displays the developed domain model.

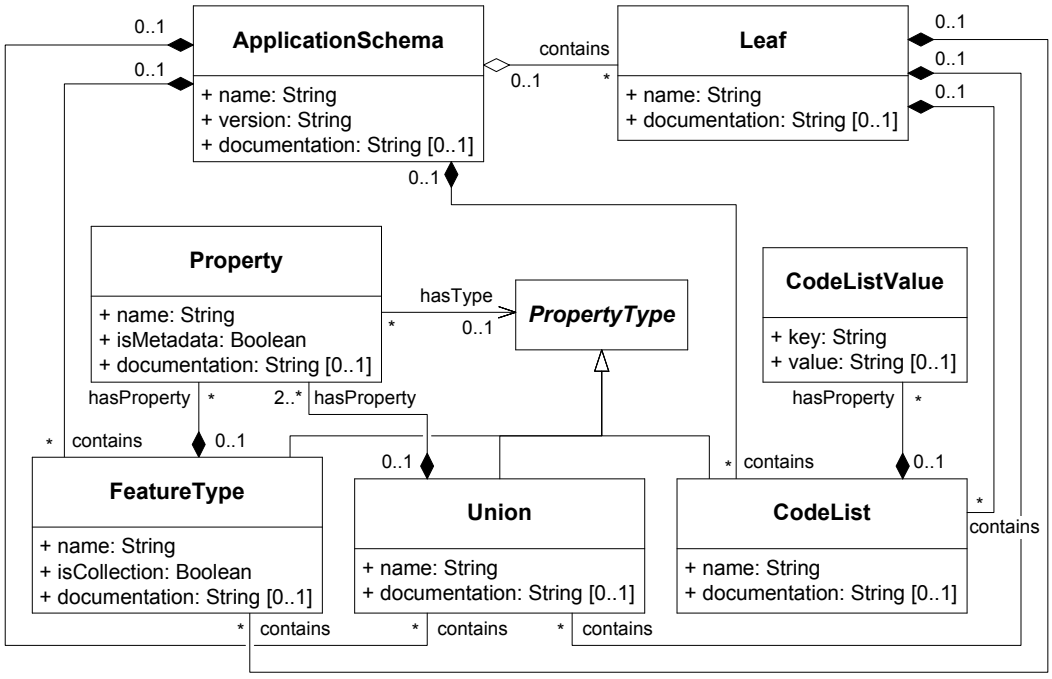


Figure 6.2: Domain model of concepts to be specified in the Core UML profile. The domain model only defines concepts which are specific to the geospatial domain, general concepts which already exist in the UML metamodel (e. g. data type and enumeration) are not included.

Based on the finding that the ISO/TS 19103 UML profile and the ISO 19109 UML profile are relevant to the entire geospatial domain, the concepts from these UML profiles should be represented in the domain model in any case. This comprises the concepts *code list*, *union* and *leaf* from the ISO/TS 19103 UML profile, which are defined in the domain model by the classes *CodeList*, *Union* and *Leaf*, as well as the concepts *application schema* and *feature type* from the ISO 19109 UML profile, which are reflected in the domain model by the classes *ApplicationSchema* and *FeatureType*. Based on the semantics provided for these concepts in the standards ISO/TS 19103 and ISO 19109, the domain model specifies, furthermore, that application schemas and leafs can be composed of feature types, code lists and unions, and that application schemas, in turn, can also aggregate leafs. Theoretically, two more concepts are of importance in geospatial data modelling – *data type* and *enumeration*. However, it was decided not to include these concepts in the domain model, as they do

not represent concepts which are specific to the geospatial domain, but general concepts which are already provided by the UML metamodel itself.

Each concept is, in addition, provided with the attribute *name* to indicate that each instance of these concepts is to be named. To decide whether further relevant attributes exist, the ISO 19136 UML profile analysed in section 5.3, page 86, is used as a basis. As part of the analysis, tables were created for each stereotype of the ISO 19136 UML profile which list the tag definitions each stereotype defines and, amongst others, indicate whether a tag definition is seen as relevant for the conceptual level or for the GML encoding. The tables show that the majority of tag definitions are encoding-specific and, thus, are not of interest for the domain model. Three tag definitions, however, were identified as being useful at the conceptual level: *documentation*, which every of the above concepts contains and which allows for adding documentation to the concepts, *version*, to declare the version of the application schema and *isMetadata*, to indicate that a specific attribute or association a feature contains represents metadata. These tag definitions are, since they appear relevant across all communities, added to the corresponding classes in the domain model as well.

In addition, also the tag definition *isCollection* from the stereotype «FeatureType» is added here. The original semantics of this tag definition specifies that by setting its value to true, 'the attribute group `gml:AggregationAttributeGroup` is added to the complex type of the feature type' (ISO 2007); the tag definition does not indicate whether a feature is to be encoded in the form of a GML feature collection. In general, every feature collection represents a feature itself. In UML modelling, a feature collection which aggregates other features is modelled identical to a feature which aggregates other features. Furthermore, the concept of the feature collection does not exist in the standard ISO 19109; it rather originates from *The OpenGIS Abstract Specification, Topic 10: Feature Collections, Version 4*, but is also seen controversially there as regards its necessity: 'perhaps Feature Collections are not needed at all' (Kottman 1999). The decision to categorise the tag definition *isCollection* not only as relevant for GML encoding, but also for conceptual modelling is based on the observation that features usually represent concrete, tangible objects of the real world, whereas feature collections rather denote imaginary concepts for grouping these concrete real-world objects. This difference is seen as relevant to be representable at the conceptual level. Examples for such feature collections are the constructs *CityModel*, *WaterNetwork* or *RoadMap*, but also *WFS feature collections* and the concept *TopicDef* from the INTERLIS UML profile (cf. section 5.5.2, page 109).

Above that, it needs to be determined, whether further concepts exist that should be included in the domain model. Since a feature type usually contains properties in the form of attributes and associations, an additional class *Property* is modelled which describes this characteristic. This class is provided with the attribute *isMetadata* from above as well as with the standard attributes *name* and *documentation*. Furthermore, each property usually has a certain type. This is indicated in the domain model by the class *PropertyType* which is modelled as an abstract superclass of the classes *FeatureType*, *Union* and *CodeList*, the concrete type of a property, thus, being either a feature type, a union or a code list (and theoretically also a data type or an enumeration). Similarly, a code list usually contains a list of values which have the form of key-value pairs. This characteristic is expressed by adding the class *CodeListValue* and the corresponding attributes *key* and *value* to the domain model.

The concepts represented in the domain model define several constraints which, however, do not require the addition of OCL constraints as they can be expressed implicitly in the domain model. The constraint that application schemas are not allowed to contain other application schemas can simply be expressed by the fact that no reflexive association is defined for the class *ApplicationSchema*. The same applies to leafs. Furthermore, a union needs to contain at least to properties to allow for

exhibiting at run-time one and only one of the properties defined. This is expressed by setting the lower bound of the multiplicity belonging to the association *hasProperty* to the value two.

The domain model exhibits to a certain extent similarities with the GFM defined in the standard ISO 19109. Also the GFM specifies concepts relevant to the geospatial domain in the form of a domain model (cf. figure A.1, page 171, which shows an extract of the GFM). However, this domain model merely focuses on the concept *FeatureType* by defining the different kinds of properties a feature can contain (operations, attributes and associations), the different types of attributes which exist (e. g. temporal attributes, spatial attributes and metadata attributes) and also that a feature can inherit from other features and that a feature can specify constraints. The domain model of the GFM does not take into account the concepts *ApplicationSchema*, *CodeList*, *Union* and *Leaf* which are defined by the domain model in this thesis. A reason for this discrepancy is that the GFM rather intends to define 'the concepts required to classify a view of the real world' (ISO 2005a) in a general way, and not to specify the concepts against the background of being able to apply them in the form of appropriate stereotypes to a UML application schema. The standard ISO 19109 defines, in fact, rules for mapping the concepts from the GFM to UML, but, since the rules do not make use of stereotypes, they also do not allow for identifying the concepts as such within a UML application schema straight away. The revision document ISO/DIS 19109:2013 started to introduce a small UML profile (cf. section 5.2, page 85), however, apart from the stereotypes «ApplicationSchema» (which actually is not represented as a concept per se in the domain model of the GFM) and «FeatureType», the UML profile does not reflect any of the concepts from the GFM, but only defines the stereotypes from the UML profile of the revision document ISO/DIS 19103:2013 anew.

6.3.2 Mapping of the domain model to the Core UML profile

After having defined the domain model, the next step is to create the Core UML profile by selecting for each concept in the domain model the most appropriate concept within the UML metamodel. In the context of the UML profile examination conducted in the previous chapter, the deficits of individual stereotypes were discussed and solutions were proposed for how to define these stereotypes compliant to the UML specification. These proposals can now be applied when defining stereotypes for the concepts from the domain model.

Figure 6.3 displays the UML profile diagram of the developed Core UML profile. The XMI structure of the Core UML profile is provided in listing C.2, page 190. The individual concepts from the domain model are realised in the Core UML profile in the following way:

- The concept *ApplicationSchema* is realised by a stereotype «ApplicationSchema» as an extension of the UML metaclass *Package* (cf. section 5.3.2.1, page 96). Since application schemas are not allowed to contain other application schemas, the stereotype receives, in addition, the OCL constraint from listing 5.5, page 97.
- The concept *Leaf* is represented by a stereotype «Leaf» which extends the UML metaclass *Package* (cf. section 5.1.2.3, page 84). Since leafs are not allowed to contain any other packages, the stereotype is complemented by the OCL constraint from listing 5.4, page 85.
- The concept *FeatureType* is realised by a stereotype «FeatureType» as an extension of the UML metaclass *Class* (cf. section 5.3.2.1, page 96).
- The concept *CodeList* is represented by a stereotype «CodeList» which extends the UML metaclass *Enumeration* (cf. section 5.1.2.1, page 82). The domain model, furthermore, contains the concept *CodeListValue* which defines the kind of values a code list can have. This concept is not mapped to a stereotype in the Core UML profile since the UML metaclass *Enumeration* provides the association

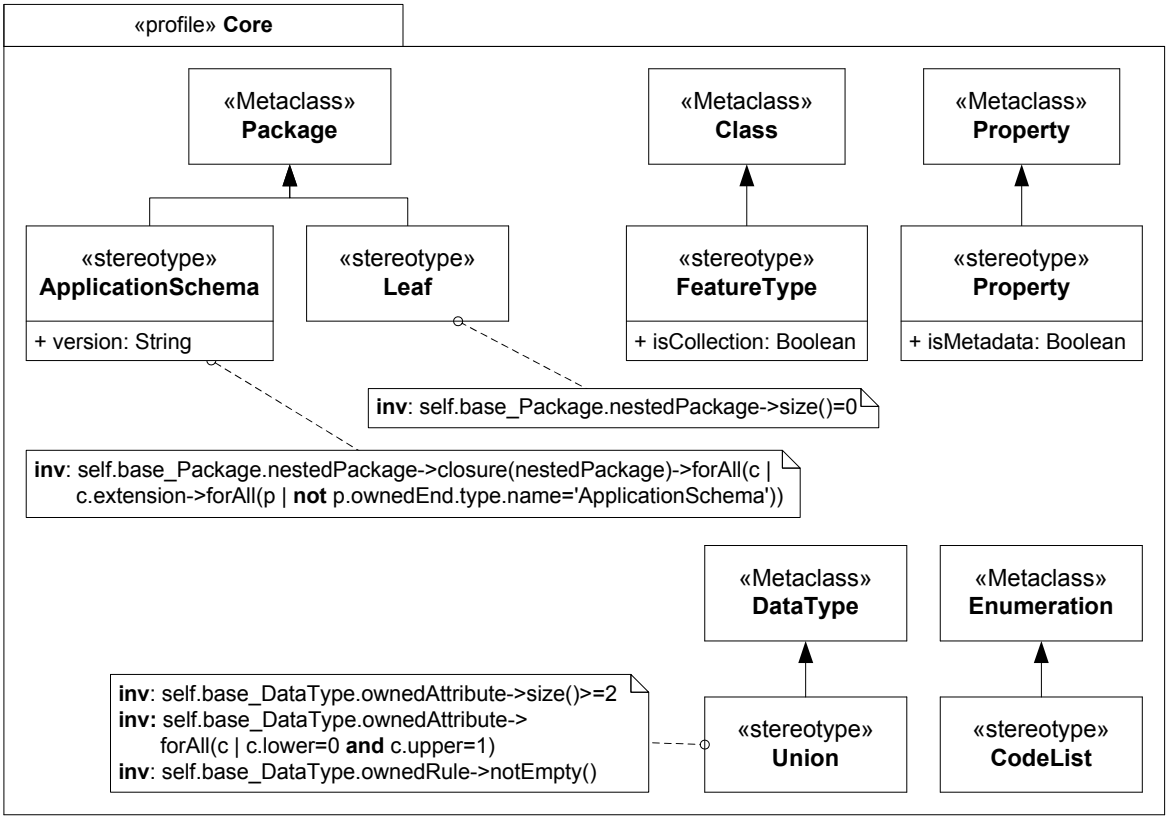


Figure 6.3: The Core UML profile

ownedLiteral for representing enumeration values, which can be used just as well for representing code list values as was proposed in section 5.1.2.1, page 82.

- The concept *Union* is realised by a stereotype «Union» which extends the UML metaclass *DataType* (cf. section 5.1.2.2, page 83). The definition that a union needs to contain at least two properties is represented by the OCL constraint from listing 5.3, page 83.

Above that, also the attributes defined in the domain model need to be realised by corresponding tag definitions. However, as can be seen in figure 6.3, not all attributes are expressed in the UML profile. The following decisions were taken:

- All UML metaclasses which are extended in the Core UML profile inherit from the UML metaclass *Element* the association *ownedComment*. This association allows for adding information to UML model elements using natural text. Since the semantics of this association corresponds to the semantics of the attribute *documentation* from the domain model, this association can be used just as well for adding documentation to UML application schemas, no specific tag definition needs to be defined for the individual stereotypes. Similarly, the UML metaclasses inherit the attribute *name* from the UML metaclass *NamedElement*. The semantics of this attribute corresponds to the semantics of the attribute *name* from the domain model; thus, no specific tag definition needs to be defined here as well.
- As regards the concepts *ApplicationSchema* and *FeatureType*, therefore, only the attributes *version* and *isCollection* remain to be specified as corresponding tag definitions, respectively.

- The attribute *isMetadata* is defined as part of the concept *Property*. To be able to add a corresponding tag definition to the Core UML profile, first a suitable stereotype needs to be specified. The UML metamodel provides the UML metaclass *Property* whose semantics is equivalent to the semantics of the domain model concept *Property*. Thus, it was decided to represent the domain model concept *Property* by a stereotype «Property» which extends the UML metaclass *Property* and to complement this stereotype with the tag definition *isMetadata*.

The associations specified in the domain model do not need to be defined explicitly in the Core UML profile since they are already reflected implicitly by the associations of those UML metaclasses the stereotypes extend.

Based on the UML profile examination conducted in the previous chapter and the general discussion of the findings therefrom in the beginning of this chapter, no other concepts are seen as fundamental to be included in the Core UML profile, to the effect that the concepts selected above represent the only core concepts required for modelling geospatial data in an adequate way. However, this Core UML profile should not be seen as a fixed, unchangeable construct for the ages. The current extent of the Core UML profile emerges from the variety of UML profiles existing in the spatial domain and the need to establish a certain order among these UML profiles which rests upon a common core. It is definitely conceivable that a certain concept, which in the beginning is part of one conceptual community UML profile only, may gradually also be included by other communities into their conceptual community UML profiles, in this way evolving in the course of the time into a concept which – due to its relevance or usefulness – is fundamental to the whole geospatial domain and, thus, into a new candidate core concept to be included in the Core UML profile.

An analogy can be drawn here to the CityGML ADE concept (cf. section 4.2.1.1, page 54, and (Gröger et al. 2012)). The CityGML ADE concept allows for enriching the CityGML model with additional information specific to a certain application, without that this information needs to be part of the CityGML specification itself. However, once this information proves to be relevant not only to a specific application, but to the modelling of 3D city and landscape models in general, the information from this ADE may become a new candidate thematic module to be included in the CityGML specification. Examples are the CityGML Tunnel ADE and the CityGML Bridge ADE (Häfele 2013) which became new thematic modules in CityGML version 2.0 and the CityGML UtilityNetwork ADE which is to become a new thematic module in CityGML version 3.0 (Löwner et al. 2014).

6.4 A framework for multi-level information integration of geospatial data based on UML profiles

The previous sections introduced concepts for the definition and use of UML profiles in geospatial data modelling which accept the variety of UML profiles existing in the geospatial domain. This section will present a framework for accepting the variety of UML profiles also within the process of information integration of geospatial data. The framework is able to solve the problems which can lead to limitations in meta-interoperability between models to which differing UML profiles are applied. This includes stereotypes which modify the semantics of the UML metamodel such that the UML models do not conform to the UML specification any more and as a result also are not fully machine-interpretable any more, but also differences regarding the UML metaclass to extend when UML versions 1 and 2 are involved.

The framework is based on the idea that a UML profile exists which serves as an intermediate step in a *multi-level information integration* process. This UML profile corresponds to the Core UML profile developed above. The idea was introduced in (Kutzner and Eisenhut 2010) and included the following short description of how the multi-level information integration could be executed: By using a 1:1 transformation tool, a source model to which UML profile x is applied is converted into a source model to which the Core UML profile is applied. This source model can then be transformed into a target model to which also the Core UML profile is applied. By using the 1:1 transformation tool again, this target model is then converted into the final target model to which the intended UML profile y is applied. The term *multi-level* refers to the fact that the transformation is not simply carried out directly between the source and target models, but that further preparatory transformations are performed on these models on account of the various UML profiles involved.

In the following, this idea will be elaborated in more detail. A framework for multi-level information integration will be introduced which operates at three layers:

- Metamodel layer: At this layer transformation definitions between UML models based on community conceptual UML profiles and UML models based on the Core UML profile are defined and executed. This layer corresponds to the layer M2 in the four-layer metamodel architecture of UML (cf. section 2.3.1, page 12).
- Model layer: At this layer transformation definitions between the source and target UML models are defined. These transformation definitions specify the actual transformation of the geospatial data. This layer is equivalent to the layer M1.
- Instance layer: This layer corresponds to the layer M0. Here the geospatial data are transformed based on the transformation definitions created at the model layer.

Since the advantages the Core UML profile brings about make it relevant enough to stand on its own and to employ it in applications beyond this framework for multi-level information integration, a suitable Core UML profile which can be used by this framework was already developed in the previous section.

6.4.1 The star-converter approach for incorporating community UML profiles in the framework

The UML profile evaluation in the previous chapter revealed that some extensions between stereotypes and UML metaclasses were mainly defined due to a syntactic compliance between the stereotypes and the UML metaclasses they extend, to the effect that the semantics of these stereotypes cannot be represented adequately and that also the semantics of the UML metamodel itself is modified, which, in turn, hinders the use of standard UML tools for defining and executing transformations between UML models which make use of these stereotypes. This problem would not exist when the stereotypes extend UML metaclasses in a semantically correct way. To solve this problem, the first part of the multi-level transformation framework will define transformations at the metamodel layer which transform semantically wrong UML models into UML models such that the semantic correctness of the stereotypes applied to these UML models as well as of the UML metamodel itself will be restored.

Central component of this transformation process is the Core UML profile; it provides semantically correct stereotypes and does not cause a semantic modification of the UML metamodel. Therefore, all source and target UML models which take part in the information integration task are first transformed into UML models to which the Core UML profile is applied. In this way, these UML models then represent their stereotypes syntactically as well as semantically in a correct way. The transformation

definitions only take into account those concepts from the source and target UML profiles which are conceptually relevant; in case the community makes use of the modular concept from section 6.2.2, page 118, this applies to all stereotypes which are part of the base UML profiles and of the conceptual community UML profiles. The transformation comprises the following steps, as is also depicted in figure 6.4:

1. Transformation definitions T_{metaSC} and T_{metaTC} are created between the source UML profile P_{source} and the Core UML profile P_{core} as well as between the target UML profile P_{target} and P_{core}, respectively. The transformation definitions are created using an appropriate editor which possibly assists the user in this task.
2. A transformation tool executes T_{metaSC}, transforming in this way the source model $M_{sourceS}$, which conforms to P_{source}, into the source model $M_{sourceC}$, which now conforms to P_{core}. Similarly, by executing T_{metaTC}, the target model $M_{targetT}$, which conforms to P_{target}, is transformed into the target model $M_{targetC}$, which now conforms to P_{core}.

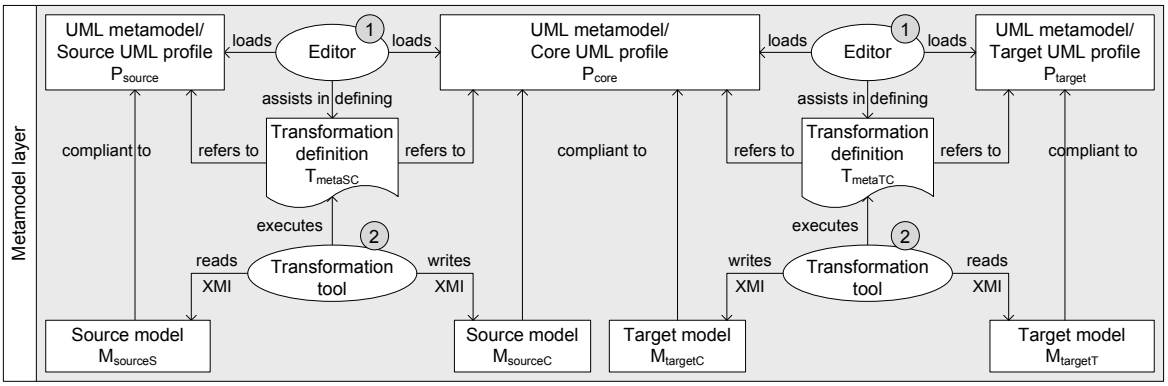
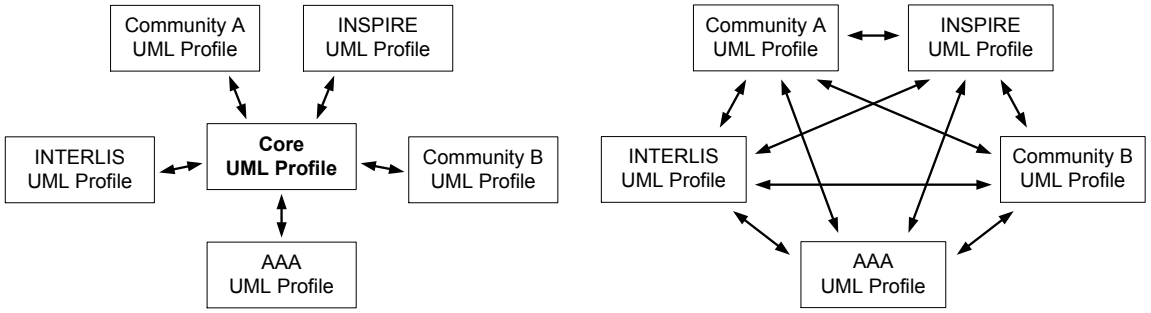


Figure 6.4: Steps conducted at the metamodel layer of the multi-level information integration framework

This transformation represents a means to overcome schematic heterogeneity, which occurs here, because the initial UML profiles and the Core UML profile use different metamodel concepts for modelling the same kind of information. This kind of transformation is a horizontal transformation which is defined at the metamodel layer between the source and target UML profiles and the Core UML profile and which is executed at the model layer on the corresponding source and target UML models. The transformation conforms to the general approach of model transformation, as is described in section 3.1.1, page 27.

The advantage of creating transformation definitions at the metamodel layer is that these transformations only need to be defined once for every pair of community UML profile and Core UML profile. Afterwards, each UML model to which one of these community UML profiles is applied can be transformed into a UML model to which now the Core UML profile is applied. This approach of transforming between different community UML profiles and the Core UML profile will in the following be referred to as *star converter approach*; it is depicted in figure 6.5(a). In general, using the star-converter approach means that for n community UML profiles n mappings need to be created, each mapping specifying bidirectionally how the concepts from one community UML profile are mappable to the concepts from the Core UML profile. In contrast, without the existence of the Core UML profile, the mappings would need to be created directly between the individual community UML profiles, which for n community UML profiles results in $n(n-1)/2$ mappings to be defined;



(a) Mapping with Core UML profile

(b) Mapping without Core UML profile

Figure 6.5: Model transformation between UML models to which different community UML profiles are applied, using (a) the star-converter approach or (b) direct mappings between the individual UML profiles

this is illustrated in figure 6.5(b). However, due to the known deficits these UML profiles exhibit, the mappings will not turn out satisfactory.

The star-converter approach bears resemblance to two other concepts. On the one hand, it is similar to the concept of model-driven transformation of geospatial data described in section 3.3.2, page 39. As part of this concept, the transformations to be executed on the geospatial data are defined one level above between the corresponding UML models. In this way, the transformations need to be defined only once and can then be applied to any data formats conforming to these UML models. On the other hand, it is also similar to the concept of using a common data transfer format for the exchanging geospatial data between different systems, instead of converting directly between the different system formats. This concept reduces for n data formats the number of format conversions to be created in each direction from $n(n-1)$ to $2n$ (Worboys and Duckham 2004). This concept is also applied by transformation tools such as the software FME which internally makes use of an FME-specific data format (cf. section 4.3.2, page 66).

6.4.2 Editor-based specification of transformation definitions at the model level

Besides the transformation definitions created at the metamodel level between different UML profiles, the general concept for multi-level information integration also requires transformation definitions to be created at the model level between specific source and target UML models. These transformation definitions specify the actual transformation of the geospatial data and conform to the concept of model-driven transformation as presented in section 3.3.2, page 39. Due to the star-converter approach, these transformation definitions are now created between source and target UML models to which the Core UML profile is applied. This guarantees that no semantic modification of the UML specification occurs any more in any of the source and target UML models. Furthermore, any standard UML tool will now be able to correctly interpret the UML models and the transformation definitions created between these UML models.

When creating normal transformations, i. e. direct transformations between source and target UML models which do not involve an intermediate step via the Core UML profile, the user bases the transformation definitions on the source and target UML models as they are. However, when a multi-

level information integration task is to be executed, also the Core UML profile has to be taken into account when creating the transformation definitions. Three different alternatives are feasible for how to create the transformation definitions when the Core UML profile needs to be considered as well. All alternatives make use of an editor supporting the user in creating the transformation definitions, for instance, by reading the source and target models and displaying them in the GUI of the editor. In this way, the user can simply select the individual model elements taking part in a specific transformation rule, without having to specify them manually.

By means of *alternative A1*, the user creates the transformation definitions compliant to the Core UML profile and also the editor outputs the transformation definitions compliant to the Core UML profile. Alternative A1 comprises the following steps, as is also depicted in figure 6.6:

1.–2. These are the two steps at the metamodel layer which were already introduced in section 6.4.1, page 130, for transforming the source and target UML models into UML models based on the Core UML profile.

3. At the model layer, the user defines the individual transformation rules between $M_{sourceC}$ and $M_{targetC}$ using an editor. To assist the user in this task, the editor needs to read and display $M_{sourceC}$ and $M_{targetC}$ beforehand.

4. The editor outputs the transformation definition $T_{modelCC}$ in a format which can later be processed by the chosen transformation tool. $T_{modelCC}$ refers to $M_{sourceC}$ and $M_{targetC}$.

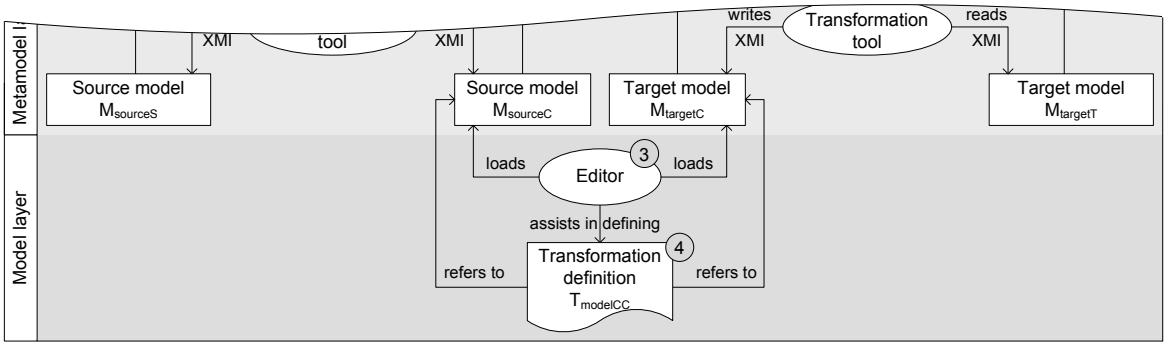


Figure 6.6: Steps conducted at the model layer of the multi-level information integration framework using alternative A1

Using *alternative A2*, the user creates the transformation definitions compliant to the Core UML profile, whereas the editor outputs the transformation definitions twice, once compliant to the Core UML profile and once compliant to the source and target UML profiles. Alternative A2 comprises the following steps, as is also depicted in figure 6.7:

1.–2. These are the two steps at the metamodel layer which were already introduced in section 6.4.1, page 130, for transforming the source and target UML models into UML models based on the Core UML profile.

3.–4. These two steps are identical to the steps 3 and 4 of alternative A1.

5. The editor knows, in addition, $M_{sourceS}$ and $M_{targetT}$ and can, thus, check and adapt the transformation rules defined by the user such that they refer to these models.

6. The editor outputs the transformation definition $T_{modelST}$, which refers to $M_{sourceS}$ and $M_{targetT}$.

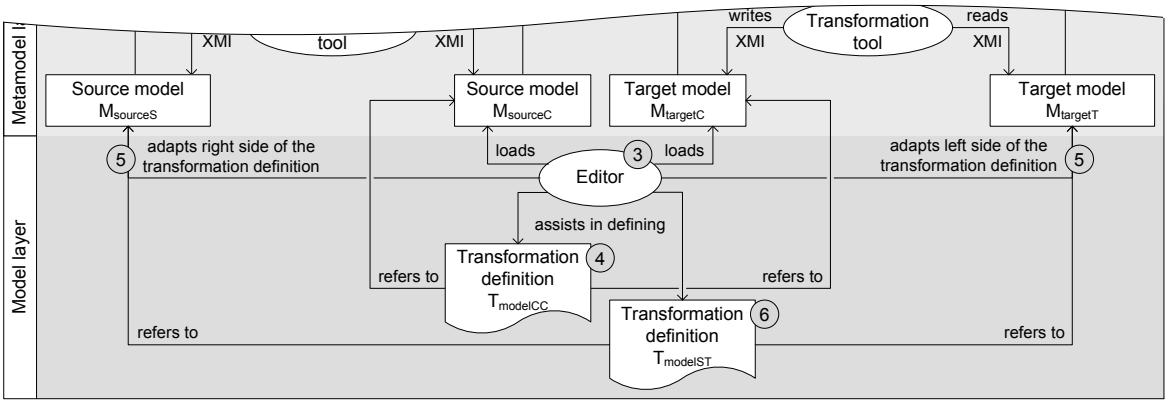


Figure 6.7: Steps conducted at the model layer of the multi-level information integration framework using alternative A2

By means of *alternative A3*, the user creates the transformation definitions compliant to the source and target UML profiles, whereas the editor outputs the transformation definitions twice, once compliant to the Core UML profile and once compliant to the source and target UML profiles. Alternative A3 comprises the following steps, as is also depicted in figure 6.8:

1.–2. These are the two steps at the metamodel layer which were already introduced in section 6.4.1, page 130, for transforming the source and target UML models into UML models based on the Core UML profile.

3. At the model layer, the user defines the individual transformation rules between $M_{sourceS}$ and $M_{targetT}$ using an editor. To assist the user in this task, the editor needs to read and display $M_{sourceS}$ and $M_{targetT}$ beforehand.
4. The editor knows, in addition, $M_{sourceC}$ and $M_{targetC}$ and can, thus, check on the basis of these models, and with reference to P_{core}, the correctness of the transformation rules defined by the user.
5. This step is identical to step 4 of alternative A1.
6. This step is identical to step 6 of alternative A2.

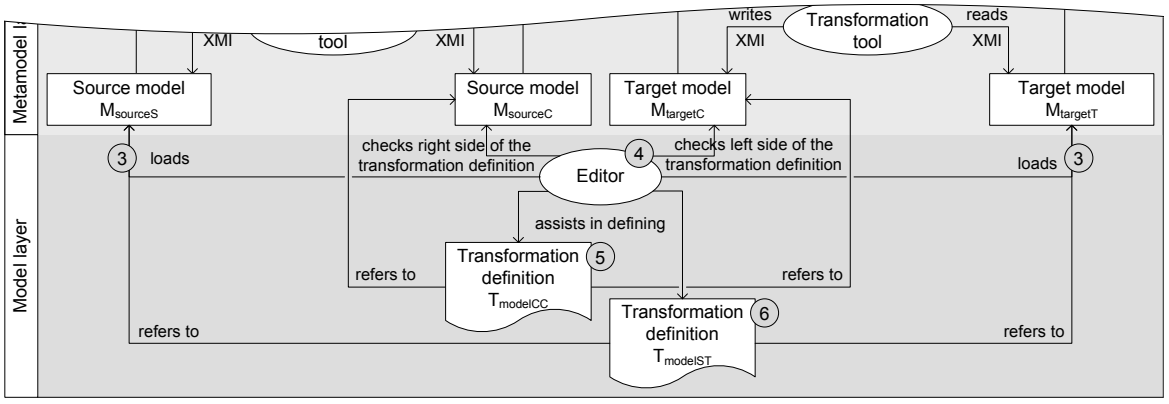


Figure 6.8: Steps conducted at the model layer of the multi-level information integration framework using alternative A3

6.4.2.1 Comparison of the alternatives

The above presented alternatives offer different advantages and disadvantages which depend on whether the user creates the transformation definitions based on the Core UML profile or based on the source and target UML profiles, and on whether the editor exports the transformation definitions solely based on the Core UML profile or, in addition, also based on the source and target UML profiles. A comparison of the alternatives is presented in table 6.2. According to this comparison, alternative A1 is the exact opposite of alternative A3 as regards their advantages and disadvantages which is due to the fact that alternative A1 operates solely within the framework of the Core UML profile, whereas alternative A3 also depends on the source and target UML profiles.

Above that, the alternatives can also be compared based on general criteria, the most important ones being *usability* and *extensibility*, but also the *implementation effort* will be addressed shortly in the following. The usability depends on whether the user can define the transformation between models which are compliant to the source and target UML profiles (i. e. $M_{sourceS}$ and $M_{targetT}$) or between models which are compliant to the Core UML profile (i. e. $M_{sourceC}$ and $M_{targetC}$). When the models are based on the Core UML profile, the user must be able to read and interpret the source and target models even when they differ from the original source and target models; this circumstance can reduce the level of usability for users which are less experienced with these altered UML models, in particular when the modifications are considerable. This criterion is also reflected in table 6.2 by the advantage/disadvantage pairs one and four.

The extensibility indicates how much effort is required for adapting the alternatives to changing UML profiles. Changes at the metamodel layer can occur, when a new concept is introduced either to a community UML profile or to the Core UML profile; in the former case, only the mapping between the modified community UML profile and the Core UML profile needs to be updated, in the latter case the mappings between all community UML profiles and the modified Core UML profile need to be adapted. These modifications affect also the editor at the model layer; when the transformation is solely created within the framework of the Core UML profile, the editor only needs to be able to interpret the modified Core UML profile, otherwise the editor also needs to be able to interpret the modified community UML profiles and the adapted transformation definitions at the metamodel layer (i. e. T_{metaSC} and T_{metaTC}). This criterion is reflected in table 6.2 by the advantage/disadvantage pairs two and three.

The implementation effort depends on whether the editor only needs to be able to interpret models compliant to the Core UML profile and, thus, only needs to generate transformation definitions (i. e. $T_{modelCC}$) which refer to these models, or whether the editor also must be able to deduce from $T_{modelCC}$ further transformation definitions (i. e. $T_{modelST}$) which refer to the original source and target models. In this case, the editor needs to know, in addition, about the original source and target models and about the transformation definitions available at the metamodel layer (i. e. T_{metaSC} and T_{metaTC}). The editor can then generate $T_{modelST}$ by adapting the left and right sides of the individual transformation rules. The implementation effort is, above that, also coupled with the extensibility criterion. The more the editor at the model layer needs to be adapted to be able to interpret changes at the metamodel layer, the higher the implementation effort is.

From a user's point of view, the best alternative is A3; it has the highest usability due to its possibility of creating the transformation based on $M_{sourceS}$ and $M_{targetT}$. Alternative A2 exhibits the same complexity regarding extensibility and implementation effort as A3, but its usability is reduced since the transformation needs to be defined based on $M_{sourceC}$ and $M_{targetC}$. Therefore, this alternative is rather not an option to take into account. In contrast, alternative A1 exhibits the lowest complexity

regarding extensibility and implementation effort since the editor only needs to be able to interpret $M_{sourceC}$ and $M_{targetC}$ and, in addition, only needs to be able to generate $T_{modelCC}$. This, however, has at the same time the effect that the usability of alternative A1 is reduced.

Table 6.2: Comparison of the individual alternatives of the multi-level information integration framework for specifying transformation definitions at the model level

Advantages (+) and disadvantages (−), listed pairwise			Alternative A1	A2	A3
1.	+	The user can define the transformation based on $M_{sourceS}$ and $M_{targetT}$.	−	−	+
	−	The user must define the transformation based on $M_{sourceC}$ and $M_{targetC}$. This may lead to difficulties, when $M_{sourceC}$ and $M_{targetC}$ differ substantially from $M_{sourceS}$ and $M_{targetT}$. In that case, the user may need to establish a relationship to the original models to be able to define the transformation correctly.			
2.	+	The star converter approach can effectively be utilised since the user defines $T_{modelCC}$ decoupled from $M_{sourceS}$ and $M_{targetT}$ and, thus, decoupled from P_{source} and P_{target}.	+	$+/-^{*}$	−
	−	The star converter approach cannot effectively be utilised, since the definition of $T_{modelCC}$ by the user does not take place decoupled from $M_{sourceS}$ and $M_{targetT}$.			
3.	+	The editor only needs to be able to interpret P_{core}.	+	−	−
	−	The editor needs to be able to interpret P_{core} as well as P_{source} and P_{target}; therefore, the editor also needs to know about T_{metaSC} and T_{metaTC}.			
4.	+	The generation of $T_{modelST}$ in addition to $T_{modelCC}$ facilitates the usage of the transformation definitions for communication purposes since $T_{modelST}$ refers to the original source and target models the user is familiar with.	−	+	+
	−	The usage of $T_{modelCC}$ for communication purposes may be hindered, when $M_{sourceC}$ and $M_{targetC}$ differ substantially from $M_{sourceS}$ and $M_{targetT}$. The user may need to establish a relationship to the original models to be able to interpret $T_{modelCC}$ correctly.			

*For generating $T_{modelST}$, the decoupling from $M_{sourceS}$ and $M_{targetT}$ can no longer be preserved, which limits the effective use of the star converter approach.

6.4.3 Transformation of the geospatial data

The last part in the multi-level information integration framework is the actual transformation of the geospatial data. The transformation is executed based on the transformation definition $T_{modelCC}$; thus, irrespective of which alternative was used for creating the transformation definition, the steps required for transforming the geospatial data are always the same.

According to the general concept of model-driven transformation of geospatial data (cf. figure 3.10, page 41), the transformation tool needs to take into account the encoding rules of the geospatial data to be transformed, in order to know according to which structure the contents have been encoded in the data formats. However, since $T_{modelCC}$ is based on P_{core}, the transformation tool can neither simply make use of the encoding rules $ER_{XsourceS}$ and $ER_{YtargetT}$ nor of the data formats $DF_{XsourceS}$ and $DF_{YsourceT}$ as they refer to P_{source} and P_{target}. Instead, encoding rules and data formats are required which refer to P_{core}. For this reason, the transformation at the instance layer comprises the following steps, as is also depicted in figure 6.9. Depending on the alternative chosen, the numbering of the steps starts either at five (alternatives A1 and A2) or at six (alternative A3):

5. Before the transformation can be executed, transformation definitions $T_{instanceSC}$ need to be defined between the encoding rule $ER_{XsourceS}$, which refers to P_{source}, and the encoding rule $ER_{XsourceC}$, which refers to P_{core}. In the same way, transformation definitions $T_{instanceTC}$ are defined between the encoding rule $ER_{YtargetT}$, which refers to P_{target}, and the encoding rule $ER_{YtargetC}$, which refers to P_{core}. These transformation definitions do not need to be created manually, they rather can be derived automatically from the transformation definitions T_{metaSC} and T_{metaTC}, respectively.
6. Based on $T_{instanceSC}$, the source data in the data format $DF_{XsourceS}$ are converted to source data in the data format $DF_{XsourceC}$; now the data structures conform to P_{core}.
7. A transformation tool executes the transformation definition $T_{modelCC}$ on the geospatial data to be transformed. The transformation tool reads the source data $DF_{XsourceC}$ and transforms these data into the target data $DF_{YtargetC}$. In doing so, the transformation tool also takes into account $ER_{XsourceC}$ and $ER_{YtargetC}$.
8. After the transformation has been executed, the transformed data $DF_{YtargetC}$ are in a last step converted into the data format $DF_{YtargetT}$ based on $T_{instanceTC}$.

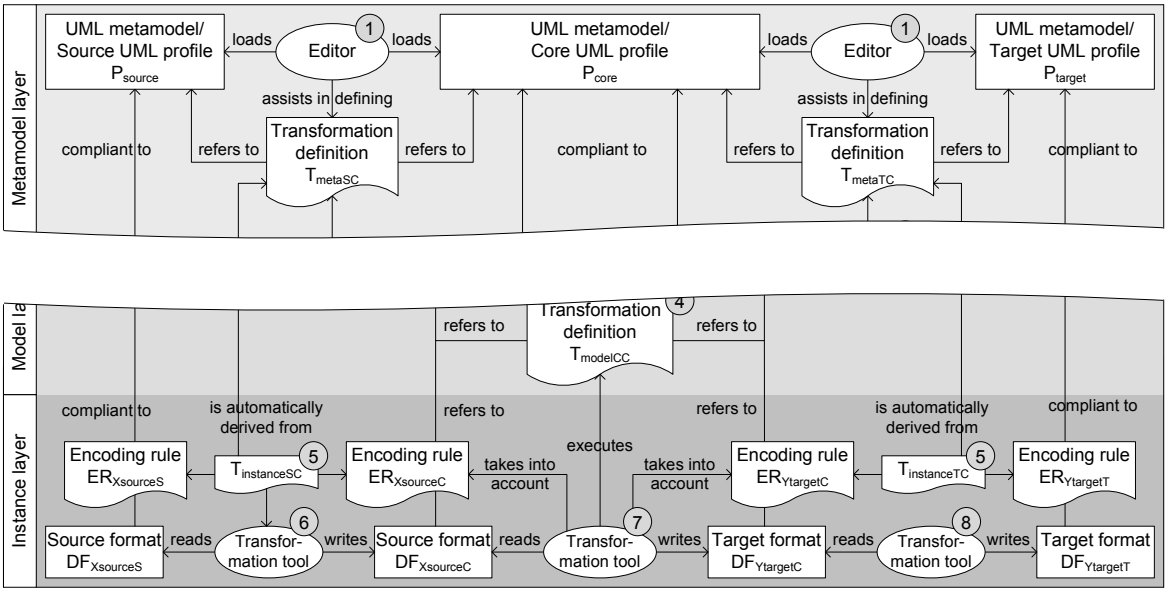


Figure 6.9: Steps conducted at the instance layer of the multi-level information integration framework

7 Proof of concept

This chapter demonstrates the feasibility of the general framework for multi-level information integration introduced in the previous chapter. The concept was implemented based on a cross-border project launched by the state and federal surveying agencies of the countries surrounding the Lake Constance. This project serves as use case for demonstrating and evaluating the applicability of the proposed framework. In the following, first the use case and relevant aspects regarding the data specifications involved in the transformation will be presented. Afterwards, the implemented workflow will be illustrated, followed by detailed explanations of the individual transformation definitions specified at the metamodel layer and at the model layer as well as of the most important aspects regarding their execution.

7.1 Use case

From 2010 to 2012 the project *Prototypical transformation of spatial data to INSPIRE in the cross-border Lake Constance region* was conducted by the Chair of Geoinformatics at the Technische Universität München. The project was launched by the state and federal surveying agencies of the countries surrounding the Lake Constance, namely Austria, Germany – represented by the two German federal states Baden-Wuerttemberg and Bavaria – and Switzerland. The project aimed at transforming the geospatial base data of topography and real estate cadastre provided by these surveying agencies to the INSPIRE Annex I themes (Schönherr et al. 2011; Kutzner, Donaubauer et al. 2014). Table 7.1 lists the data specifications[1] based on which the surveying agencies provided geospatial data for the project.

Table 7.1: Data specifications used in the project and transformations defined in the prototypical implementation of the multi-level information integration framework

Country	Topography	Cadastre	Transformations defined in the multi-level information integration task
Austria	DLM	DKM	DKM → INSPIRE CP
Baden-Wuerttemberg	ATKIS Base DLM	ALKIS	ATKIS/ALKIS → INSPIRE CP/AU
Bavaria	ATKIS Base DLM	DFK	DFK → INSPIRE CP/AU
Switzerland	TLM	MOpublic	MOpublic → INSPIRE CP/AU

The first part of the project focused on applying a format-schema-driven transformation on these geospatial data using existing commercial transformation tools and on offering the transformed geospatial data afterwards via suitable view and download services. This part was carried out by the company AED-SICAD AG, a project partner of the Chair of Geoinformatics within the project. The software FUSION Data Service (FDS) was used for this transformation; FDS is based on the software

[1] Please refer to the list of acronyms for the meaning of the acronyms denoting the data specifications.

FME (cf. section 4.3.2, page 66), extending FME by a so-called Semantic Mapper module which was developed by AED-SICAD AG (Lill et al. 2011; Kutzner, Schilcher et al. 2012). Within this part, the geospatial data were transformed into all themes of INSPIRE Annex I. The second part of the project, in contrast, focused on a model-driven transformation of the geospatial data by applying the multi-level information integration framework presented in the previous chapter. Since this part rather aimed at demonstrating the feasibility of the introduced framework, the transformations were only realised for the INSPIRE themes Cadastral Parcels (CP) and Administrative Units (AU). The specific transformations considered are stated as well in table 7.1.

As regards the data specifications involved in the second part of the project, they exhibit only in part formal data models specified using a modelling language; some data specifications are defined by means of feature catalogues or other specification documents using natural language only. Table 7.2 provides an overview, whether a publicly available formal data model exists for the data specification and, if this is the case, which modelling language was used for defining the data model. The table, furthermore, provides information whether a UML profile is applied to the data model and in which data formats the geospatial data were provided by the surveying agencies. Besides the data specifications representing the source side of the transformation, this overview also includes the target side, i. e. the INSPIRE data specifications.

Table 7.2: Formal data models and data formats of the data specifications used in the prototypical implementation of the multi-level information integration framework

Data specification	Formal data model	Modelling language	UML profile	Data format
DKM	no	–	–	Shape, ASCII
ATKIS Base DLM/ ALKIS	yes, the AAA model	UML 1.4.2	AAA UML profile	NAS
DFK	no	–	–	Shape
MOpublic	yes	INTERLIS	INTERLIS UML profile	INTERLIS 2
INSPIRE CP/AU	yes	UML 2.1	INSPIRE UML profile	INSPIRE GML

7.2 System architecture of the prototypically implemented framework

This section describes the workflow which was composed for transforming the geospatial data to INSPIRE based on the multi-level information integration framework introduced in the previous chapter. Furthermore, the software tools used for realising the individual steps of the workflow are shortly summarised.

7.2.1 Workflow of the multi-level information integration framework

Since the aim of the second part of the project was to demonstrate the feasibility of the multi-level information integration approach, alternative A1 (cf. section 6.4.2, page 132) was chosen for defining and implementing the multi-level information integration concept, as it has the lowest implementation effort. Furthermore, it seemed reasonable that the user defines the transformation rules based on the

Core UML profile; on the one hand, the Core UML profile is slim and, therefore, easy to understand, on the other hand, the concepts of the Core UML profile represent an intersection of those concepts relevant across all (evaluated) communities, which, thus, are available in all (evaluated) UML profiles, leading to the conclusion that the user is familiar with them anyway.

Another objective of the second part of the project was to reuse – and, if required, to extend – the developments from the mdWFS project (cf. section 4.3.3, page 67), in particular the transformation language UMLT, including the UMLT editor, as well as the FME extensions XMI Reader and UMLTApplier transformer for executing the UMLT transformations. Thus, the transformation definitions at the model layer were created using UMLT. The transformation definitions at the metamodel layer were specified using ATL (cf. section 7.3.4, page 151, for an explanation why ATL was chosen). Based on these conditions, the complete workflow for transforming the source geospatial data to the INSPIRE themes CP and AU was composed as depicted in figure 7.1.

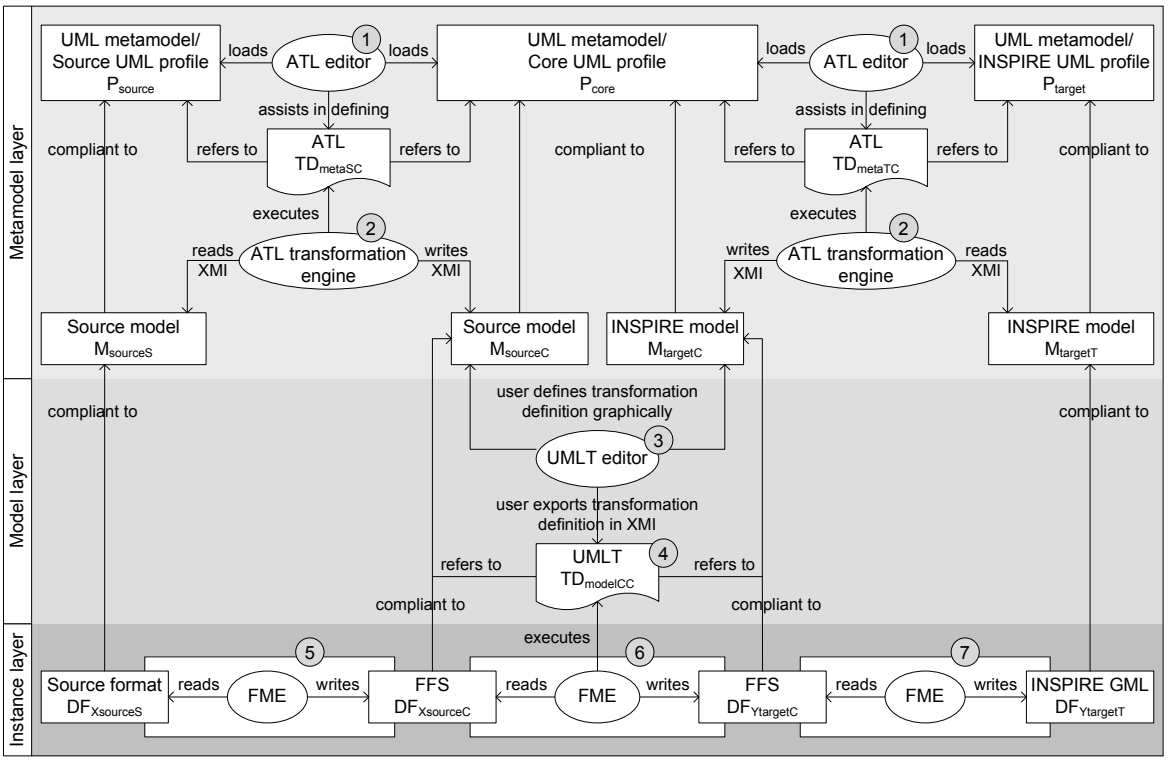


Figure 7.1: Workflow of the multi-level information integration framework prototypically implemented based on alternative A1

The workflow comprises the following steps:

1. At the metamodel layer, transformation definitions TD_{metaSC} and TD_{metaTC} are specified between the source UML profiles P_{source} and the Core UML profile P_{core} as well as between the target UML profile P_{target}, namely the INSPIRE UML profile, and P_{core} using the transformation language ATL. TD_{metaSC} and TD_{metaTC} are created by means of the ATL editor, which is part of the ATL plug-in for Eclipse. The ATL editor offers content assist for EMF-based metamodels by specifying at the beginning of the ATL transformation definition the metamodel to be loaded by the ATL editor. In this way, UML2, the EMF-based implementation of the UML metamodel for

the Eclipse platform, can be loaded by the ATL editor as well as the UML profiles used in this prototypical implementation.

2. TD_{metaSC} and TD_{metaTC} are executed using the ATL transformation engine, which is part of the ATL plug-in for Eclipse, too. The source and target UML models $M_{sourceS}$ and $M_{targetT}$ are provided to the engine as XMI documents. The engine reads these models, transforms them into models $M_{sourceC}$ and $M_{targetC}$ to which P_{core} is applied and writes the transformed models as XMI documents. To be able to consider the UML profiles during the transformation process, the engine also needs to read the involved UML profiles (not shown in the diagram).
3. At the model layer, transformation definitions $TD_{modelCC}$ are specified between $M_{sourceC}$ and $M_{targetC}$ using the transformation language UMLT. The general concept developed in section 6.4.2, page 132, assumes that this task makes use of an editor which assists the user in creating $TD_{modelCC}$. The UMLT editor from the mdWFS project is employed for this purpose.
4. After having created $TD_{modelCC}$, the user exports these definitions from the UMLT editor as XMI documents.

The next three steps in the workflow are implemented by means of FME. The detailed set-up of the FME workspaces created for executing the steps will be explained in section 7.5.2, page 156.

5. Before the geospatial data can be transformed, their data structures have to be converted to data structures conforming to $M_{sourceC}$. This conversion is theoretically based on transformation definitions located at the level of the encoding rules by automatically deriving them from TD_{metaSC} and TD_{metaTC} (cf. section 6.4.3, page 136). However, since the encoding rules for the source models used in the project do not exist in machine-interpretable form, a workaround needed to be devised. Within this prototypical implementation, step five was solved manually by creating so-called encoding workspaces using FME which read the geospatial data $DF_{XsourceS}$ provided in the data formats listed in table 7.2, convert them and write them to the FME-specific data format FFS (FME Feature Store), i. e. $DF_{XsourceC}$.
6. This step actually transforms the geospatial data. The transformations are executed by means of a so-called transformation workspace which is created using FME, too. The workspace reads $DF_{XsourceC}$, executes $TD_{modelCC}$, i. e. the UMLT definitions which were exported in step four as XMI documents, and writes the transformed geospatial data $DF_{YtargetC}$ again in the FFS format such that the data conform to the INSPIRE data model $M_{targetC}$. This step represents an automatic execution of transformation definitions defined at the conceptual model layer on geospatial data located at the instance layer.
7. The last step in this workflow was realised again as workaround in the form of an FME encoding workspace since also for the INSPIRE model the encoding rules do not exist in machine-interpretable form. The workspace reads $DF_{YtargetC}$, converts the data and writes them to the final INSPIRE GML format $DF_{YtargetT}$.

The above described workflow can be used in exactly this way for transforming the data specifications ATKIS Base DLM, ALKIS and MOpublic to the INSPIRE data specifications since formal data models exist for them which allow for defining the transformation definitions at the model layer using UMLT. As table 7.2 states, also the data specifications DKM and DFK are to be transformed to INSPIRE. However, no formal data models are available for these data specifications. To be able to apply the multi-level information integration approach to the corresponding geospatial data nevertheless, suitable data models were specifically defined for them using UML 2.1. The contents and structures of these data models were derived from the available feature catalogues and specification documents as well as from the geospatial data itself. The definition of these data models made aware of the general problems which can occur when developing data models based on the underlying data

formats and related specification documents (cf. section 4.2.3.1, page 61). Furthermore, it was decided that directly the Core UML profile is to be applied to these data models since, on the one hand, the data specifications did not exhibit any concepts beyond those provided by the Core UML profile; on the other hand, applying the Core UML profile to them guarantees a semantic correctness of the stereotypes defined therein.

7.2.2 Technologies employed for implementing the framework

This section summarises the software tools employed for implementing the workflow. Since most of the implementation was conducted in 2012 already, this summary reflects the software versions which were up-to-date at that time.

The XMI Reader, the UMLTApplier and the UMLT editor are developments which were reused from the mdWFS project. For being able to use them within this workflow they needed to be extended regarding certain aspects which was done using *Eclipse Indigo* SR2 (Eclipse version 3.7.2) in the form of the Eclipse Modeling Tools package. This package already contains several required plug-ins, in particular *Eclipse EMF* version 2.7.0 (The Eclipse Foundation 2015c; Steinberg et al. 2009) and *Eclipse GMF* (The Eclipse Foundation 2015d) version 1.5.0 based on which the UMLT editor was developed, and *Eclipse UML2* version 3.2.1 (The Eclipse Foundation 2015e) for processing the UML models, UML profiles and UMLT definitions in the form of XMI documents within the XMI Reader and the UMLTApplier.

The transformation definitions at the metamodel layer were specified and executed using the *Eclipse ATL* plug-in version 3.2.1 (The Eclipse Foundation 2015a). As will be described in section 7.3.4, page 151, in the beginning also QVT was considered as suitable language for specifying the transformation definitions. Several tests were conducted using the Eclipse-based software tool *mediniQVT* version 1.6.0 (KPIT medini Technologies 2012) as well as the *Eclipse QVTo* plug-in version 3.1.0 (The Eclipse Foundation 2015f).

The AAA UML model (version 6.0.1) and the INSPIRE UML model (revision 937) were available as Enterprise Architect projects from the corresponding web sites (AdV 2015) and (JRC 2015). Enterprise Architect is the UML tool predominantly used in the geospatial domain. Thus, also the UML models for the data specifications DKM and DFK were developed using *Enterprise Architect* version 10 (Sparx Systems 2015). The UML models were exported from Enterprise Architect as XMI 2.1 documents. To be able to process these XMI documents with Eclipse UML2, the software *MagicDraw* version 17.0.1 (No Magic 2015) was used in a pre-processing step as will be explained in section 7.5.1, page 155. The AAA and the INSPIRE UML profiles as well as the Core UML profile were (re-)modelled using MagicDraw and exported as Eclipse XMI documents. Table 7.2 states that the MOpublic model is actually modelled using the modelling language INTERLIS. To be able to include this model in the framework nevertheless, a UML-based MOpublic model and also the INTERLIS UML profile were provided to the project directly as Eclipse XMI documents by swisstopo.

The encoding and transformation workspaces in the steps five, six and seven were defined and executed using *FME Desktop 2012* SP2 (Safe Software 2015a).

7.3 Specification of transformations definitions at the metamodel level

This section describes in more detail step one of the workflow, i. e. the specification of transformation definitions at the metamodel layer defining the transformation of those UML models to which community-specific UML profiles are applied into UML models to which the Core UML profile is applied. The transformation definitions need to specify, on the one hand, mappings of the UML metaclasses and, on the other hand, mappings of the stereotypes between the community-specific UML profiles and the Core UML profile. The stereotype mappings also need to include mappings between the corresponding tag definitions; these mappings only need to consider tag definitions from the community-specific UML profiles which are identified as being relevant at the conceptual level, encoding-specific tag definitions are not of relevance here.

In the following, mappings are defined for the INSPIRE UML profile, the AAA UML profile and the INTERLIS UML profile. These three UML profiles are listed in table 7.2, page 140, as the UML profiles applied to the data specifications used in the project. Above that, these UML profiles were also classified as community conceptual UML profiles (cf. table 6.1, page 206), which means that they are applied to conceptual UML models at the PIM level and, therefore, are well-suited for being used in the star-converter approach. The mappings provided here can be used as guidelines and examples for other UML profiles to which the multi-level information integration concept is to be applied as well.

The mappings are provided in the form of simple mapping tables. The left side of the table specifies the target of the mapping, i. e. the Core UML profile, the right side specifies the source, i. e. one of the above mentioned UML profiles. Each side of the table contains two columns, one for the UML metaclass and one for the corresponding stereotype to be mapped. Theoretically, also a column for the tag definitions to be mapped should be part of the left and right side of the table. However, for purposes of clarity regarding the page layout, and also because the mapping only affects very few tag definitions, this column is not included here; instead, the mapping is defined textually.

Some of the mappings between UML metaclasses require more detailed specifications which take into account individual features of these UML metaclasses as well. In UML, features represent characteristics of a classifier (ISO 2012b), the features considered here being attributes and associations. Additional mapping tables are provided for these specifications, the left side of the table stating the target UML metaclass, the right side stating the source UML metaclass. Each side of these tables contains three columns, one for the UML metaclass, one for the feature and one for the type of the feature[2].

As mentioned in the workflow description, the mappings are to be implemented using the transformation language ATL. Therefore, this section will also point out the reasons having led to the decision to implement the mappings using ATL and will exemplary provide the ATL code for the INSPIRE UML profile to Core UML profile mapping.

7.3.1 Mapping of the INSPIRE UML profile to the Core UML profile

The mapping between the INSPIRE UML profile and the Core UML profile is based on the current INSPIRE UML profile, as discussed in section 5.4.1, page 101, and displayed in figure C.6, page 188.

[2] Some of the UML metaclasses and features discussed here are represented in the UML metamodel diagrams in appendix A.2. If required, please refer to these diagrams for a better understanding of the mapping tables. The feature *name*, however, is not visible there, the UML metaclasses inherit this feature from the UML metaclass *NamedElement*.

Table 7.3: Metamodel mapping of UML metaclasses and stereotypes between the INSPIRE UML profile and the Core UML profile

Core UML profile Stereotype	UML metaclass	INSPIRE UML profile Stereotype	UML metaclass
«ApplicationSchema»	Package	«applicationSchema»	Package
–	Package	any or no stereotype	Package
«Leaf»	Package	«leaf»	Package
«FeatureType»	Class	«featureType»	Class
«FeatureType»	Class	«placeholder»	Class
–	Class	no stereotype	Class
«Type»*	Class	«type»	Class
«Union»	DataType	«union»	Class
«CodeList»	Enumeration	«codeList»	Class
–	Enumeration	«enumeration»	Class
–	DataType	«dataType»	Class
–	PackageImport	«import»	Dependency
Mapped to intermediate classes		«voidable»	Attribute, AssociationEnd
–	Property	«lifeCycleInfo»	Attribute, AssociationEnd
–	Property	«version»	AssociationEnd
«Property»	Property	no stereotype	Attribute, AssociationEnd

*The stereotype «Type» is not part of the Core UML profile, it actually belongs to the UML StandardProfileL2

Table 7.3 shows in which way the UML metaclasses and stereotypes from the INSPIRE UML profile are mapped to suitable UML metaclasses and stereotypes from the Core UML profile.

It can be noticed that the mapping is most straightforward for the INSPIRE stereotypes «applicationSchema», «leaf» and «featureType». These stereotypes as well as the UML metaclasses they extend can simply be mapped to identical stereotypes and UML metaclasses in the Core UML profile without any further effort.

Similarly, the stereotype «codeList» can be mapped to an identical stereotype in the Core UML profile, however, the UML metaclasses differ in this mapping. The INSPIRE UML profile uses the UML metaclass *Class* for this stereotype, whereas the Core UML profile uses the UML metaclass *Enumeration*. Since the UML metaclasses *Class* and *Enumeration* exhibit differing features, a more detailed mapping is required between them which is shown in table 7.4. This table specifies a mapping between the attribute *name* belonging to the UML metaclass *Class* and an identical attribute *name* belonging to the UML metaclass *Enumeration*. This *name* attribute holds the name of the code list. The UML metaclass *Class*, furthermore, contains the association *ownedAttribute* of the type *Property* which holds the key-value pairs of the code list. This association is mapped to the association *ownedLiteral* of the type *EnumerationLiteral* which allows for specifying the key-value pairs as enumeration literals. This is stated in the second mapping. The *name* attribute of the UML metaclass *Property* holds the code list value and the *defaultValue* association holds the code list key. Value and key are concatenated in the form of *value=key* as proposed in section 5.1.2.1, page 82, and are mapped to the *name* attribute of the UML metaclass *EnumerationLiteral*, each *name* attribute representing a specific literal value.

Table 7.4: Metamodel mapping between the UML metaclasses *Class* and *Enumeration* required for the stereotypes «codeList» and «enumeration» from the INSPIRE UML profile

Core UML profile			INSPIRE UML profile		
UML metaclass	Feature	Type	UML metaclass	Feature	Type
Enumeration	name	String	Class	name	String
	ownedLiteral	EnumerationLiteral		ownedAttribute	Property
EnumerationLiteral	name	String	Property	name	String
				= defaultValue.value	String

A more detailed mapping also needs to be defined for the stereotype «union» which extends the UML metaclass *Class* in the INSPIRE UML profile, but the UML metaclass *DataType* in the Core UML profile. This mapping is shown in table 7.5. It specifies that the attribute *name* belonging to the UML metaclass *Class* is mapped to an identical attribute *name* belonging to the UML metaclass *DataType*. This *name* attribute holds the name of the union. Furthermore, the association *ownedAttribute* belonging to the UML metaclass *Class* holds the attributes of the union; this association is mapped to an equally named association belonging to the metaclass *DataType*. Both associations are of the type *Property*, thus, this mapping does not need to be specified any further.

Table 7.5: Metamodel mapping between the UML metaclasses *Class* and *DataType* required for the stereotypes «union» and «dataType» from the INSPIRE UML profile

Core UML profile			INSPIRE UML profile		
UML metaclass	Feature	Type	UML metaclass	Feature	Type
DataType	name	String	Class	name	String
	ownedAttribute	Property		ownedAttribute	Property

The stereotypes «enumeration» and «dataType», both extending the UML metaclass *Class* in the INSPIRE UML profile, are represented as keywords in the Core UML profile. Therefore, the UML metaclass *Class* is mapped once to the UML metaclass *Enumeration* and once to the UML metaclass *DataType*, using the mappings from the tables 7.4 and 7.5 again. The stereotypes themselves are not mapped.

The stereotype «import», which extends the UML metaclass *Dependency* in the INSPIRE UML profile, is represented as keyword in the Core UML profile, too. This UML metaclass is mapped to the UML metaclass *PackageImport* in the Core UML profile as shown in table 7.6. This mapping represents, in particular, a rule for transforming between UML 1 and UML 2. In UML 1, the UML metaclass *Dependency* serves in defining package imports, whereas UML 2 introduces the UML metaclass *PackageImport* for this task (cf. section 5.3.1.2, page 88). Since the INSPIRE UML model is defined directly using UML 2, this mapping should not be required when the modeller uses the correct UML metaclass from the beginning. In practice, however, the currently available revision 4618 of the INSPIRE UML model still exhibits cases, where the UML metaclass *Dependency* is used for defining package imports (JRC 2015).

Similarly, the table defines a mapping between the UML metaclasses *Attribute* and *AssociationEnd* for all those attributes and associations without stereotype to the UML metaclass *Property* which

Table 7.6: Metamodel mapping between the UML metaclasses *Dependency* and *PackageImport* required for the stereotype «import» from the INSPIRE UML profile

Core UML profile			INSPIRE UML profile		
UML metaclass	Feature	Type	UML metaclass	Feature	Type
PackageImport	importingNamespace importedPackage	Namespace Package	Dependency	client.namespace supplier	Namespace NamedElement

is extended by the stereotype «Property». The stereotype «Property» was introduced in the Core UML profile to be able to add the tag definition *isMetadata* to attributes and associations. As regards the UML metaclasses, also this mapping represents a rule for transforming between UML 1 and UML 2; the UML metaclasses *Attribute* and *AssociationEnd* only exist in UML 1, in UML 2 they were replaced by the UML metaclass *Property*. Since the INSPIRE UML model is defined directly using UML 2, the attributes and associations are already modelled correctly using the UML metaclass *Property*, which is extended through the mapping by the stereotype «Property».

The stereotype «type» from the INSPIRE UML profile is actually already predefined in the UML 2 specification as part of the UML StandardProfileL2. Since their semantics are identical and since also their UML metaclasses conform, the stereotype «type» from the INSPIRE UML profile is simply mapped to the stereotype «type» from the UML StandardProfileL2.

The stereotype «placeholder» is an INSPIRE-specific concept which does not exist in the Core UML profile. Since a placeholder represents a feature type to be defined in the future (cf. section 5.4.1.1, page 101), it was decided to map this concept to the stereotype «FeatureType» in the Core UML profile.

The stereotype «voidable» is an INSPIRE-specific concept as well. It cannot be mapped to the Core UML profile as no corresponding concept exists there. However, section 5.4.1.2, page 102, pointed out that this concept causes a semantic modification of the UML specification. Therefore, even if the stereotype itself is not mappable, a conversion of the UML model has to be conducted to eliminate the still existing semantic modification. Two possibilities are feasible for the conversion. The first possibility is to not map the stereotype «voidable», but to keep the properties to which the stereotype was applied as they are otherwise, and, in addition, to map the code list *VoidReasonValue* (cf. figure 5.11, page 103) as defined in table 7.4. The second possibility is to not map the stereotype «voidable» as well, but to apply at the same time the proposal from section 5.4.2, page 104. In this way, the voidable concept can implicitly be preserved in the Core UML profile. The mapping in table 7.3 specifies the second option.

Above that, also the stereotypes «lifeCycleInfo» and «version» are INSPIRE-specific concepts. For them, and also for the any or no stereotype concepts, only a mapping to the corresponding UML metaclasses is defined, since no corresponding concepts exist in the Core UML profile (except for the stereotype «Property», as explained above).

As regards the tag definitions, the stereotype «ApplicationSchema» from the INSPIRE UML profile contains the mappable tag definition *version*, which is mapped to the equally named tag definition in the Core UML profile. Furthermore, attributes and associations without stereotype have in the INSPIRE UML model the tag definition *isMetadata* which is mapped to the equally named tag definition from the stereotype «Property». Most stereotypes also contain the tag definition *documentation*; it is mapped in accordance with section 6.3.2, page 127, to the association *ownedComment*, which

every UML metaclass contains. The stereotype «FeatureType» contains the tag definition *isCollection*, whose semantics in the INSPIRE UML profile differs, however, from the semantics in the Core UML profile, thus, the values cannot be mapped. All other tag definitions are not relevant at the conceptual level.

7.3.2 Mapping of the AAA UML profile to the Core UML profile

In the same way, mappings can be defined between the AAA UML profile and the Core UML profile. Table 7.7 lists these mappings. As mentioned in section 5.5.1, page 108, the AAA documents do not give a very clear picture of the structure and the extent of the AAA UML profile; for this reason, the stereotypes and tag definitions which seem to be part of the AAA UML profile were only textually summarised in that section. The mappings defined here are based on this summary and reflect the stereotypes of the AAA UML profile as understood from the AAA documents.

Table 7.7: Metamodel mapping of UML metaclasses and stereotypes between the AAA UML profile and the Core UML profile

Core UML profile Stereotype	UML metaclass	AAA UML profile Stereotype	UML metaclass
«ApplicationSchema»	Package	«applicationSchema»	Package
–	Package	«schema»	Package
–	Package	any or no stereotype	Package
«Leaf»	Package	«leaf»	Package
«FeatureType»	Class	«featureType»	Class
–	Class	no stereotype	Class
«Type»*	Class	«type»	Class
«Union»	DataType	«union»	Class
«CodeList»	Enumeration	«codeList»	Class
–	Enumeration	«enumeration»	Class
–	DataType	«dataType»	Class
–	PackageImport	«import»	Dependency
«Property»	Property	no stereotype	Attribute, AssociationEnd
–	–	«Request»	not specified
–	–	«Response»	not specified

*The stereotype «Type» is not part of the Core UML profile, it actually belongs to the UML StandardProfileL2

The mappings for the stereotypes «applicationSchema», «leaf», «featureType», «type», «union», «codeList», «enumeration», «dataType» and «import», for the any or no stereotype concepts and for the UML metaclasses are equivalent to the mappings defined for the INSPIRE UML profile the previous section and, thus, are not explained here again. For the stereotype «schema», which is used in the AAA Enterprise Architect project, no meaning is provided in the AAA documents; therefore, the only definable mapping is between its UML metaclass *Package* and an identical UML metaclass in the Core UML profile. As regards the stereotypes «Request» and «Response», neither do the AAA documents specify suitable UML metaclasses, nor it is possible to deduce the UML metaclasses from the AAA Enterprise Architect project since the stereotypes are not used there. Above that, their

meaning does not unambiguously allow for concluding whether they should rather be part of the AAA conceptual UML profile or of an AAA encoding UML profile. For these reasons, the two stereotypes are listed in the table, but are not included in the mapping.

According to the GeoInfoDoc document, the AAA application schema uses all tag definitions from the standard ISO 19136 Annex E. This means that the AAA UML profile contains, amongst others, the tag definitions *version*, *isCollection* and *isMetadata*, which are at the same time also part of the Core UML profile. This requires pairwise mappings to be added to the stereotypes the tag definitions are part of, namely a mapping of *version* to the stereotype «ApplicationSchema», a mapping of *isCollection* to the stereotype «FeatureType» and a mapping of *isMetadata* to the stereotype «Property». The tag definition *documentation*, which is also part of the AAA UML profile, is mapped to the UML metaclass association *ownedComment* as was explained in section 6.3.2, page 127. In addition, the stereotypes of the AAA UML profile specify several AAA-specific tag definitions. However, they rather seem suitable to a AAA community encoding UML profile and are, thus, not considered here any further.

The mapping of the AAA UML profile to the Core UML profile is altogether quite similar to the mapping of the INSPIRE UML profile to the Core UML profile since many concepts are identical. Therefore, also other UML profiles to which the star-converter approach is applied and whose concepts overlap comparably, will result in more or less analogous mappings.

7.3.3 Mapping of the INTERLIS UML profile to the Core UML profile

In contrast to the previous two mappings, the following mapping between the INTERLIS UML profile and the Core UML profile demonstrates that the star-converter approach also withstands UML profiles which are not defined in the context of the ISO 191xx series of geographic information standards.

INTERLIS can be regarded as a self-contained approach which exists besides the ISO-related approaches. The concepts defined by the INTERLIS UML profile look at first glance quite different from the concepts defined by the standards ISO/TS 19103 and ISO 19109, not least due to a different naming. Upon closer examination, however, many correspondences become apparent which makes a mapping between the concepts of the INTERLIS UML profile and the concepts of the Core UML profile fully feasible. Table 7.8 displays the defined mappings[3].

The INTERLIS UML profile provides the stereotype «ModelDef» which is used to denote application schemas. Since application schemas are represented in UML-based INTERLIS models as UML packages as well, the stereotype can smoothly be mapped to the stereotype «ApplicationSchema» in the Core UML profile.

The stereotype «TopicDef» serves in grouping the aspects modelled in an INTERLIS application schema thematically into individual data baskets. The INTERLIS UML profile defines the stereotype as extension of the UML metaclass *Package* which requires the data baskets to be represented as individual UML packages within an INTERLIS application schema. In the terminology of the ISO-related approaches, however, these data baskets rather correspond to feature collections. This

[3]It has to be noted here that the INTERLIS UML profile was specifically provided by swisstopo for being able to compare the INTERLIS concepts with concepts of already existing UML profiles in the geospatial domain (cf. section 5.5.2, page 109, and (Kutzner and Eisenhut 2010)). Furthermore, the INTERLIS UML profile is required for being able to include INTERLIS models in the multi-level information integration framework, which, in turn, requires the models themselves to be modelled using UML and not INTERLIS. The INTERLIS UML profile is normally not in use in geospatial data modelling in Switzerland since INTERLIS models are defined using the modelling language INTERLIS and, thus, do not require an INTERLIS UML profile applied to them. Also, the UML-based MOpublic model to which the INSPIRE UML profile is applied was specifically provided by swisstopo for use in this project.

Table 7.8: Metamodel mapping of UML metaclasses and stereotypes between the INTERLIS UML profile and the Core UML profile

Core UML profile Stereotype	UML metaclass	INTERLIS UML profile Stereotype	UML metaclass
«ApplicationSchema»	Package	«ModelDef»	Package
«Leaf»	Package	–	–
«FeatureType»	Class		
with tag definition isCollection=true		«TopicDef»	Package
with tag definition isCollection=false		–	Class
«Union»	DataType	–	–
«CodeList»	Enumeration	–	–
–	Enumeration	–	Enumeration
–	DataType	–	DataType
–	DataType	«LineFormTypeDef»	DataType
–	DataType	«UnitDef»	DataType
–	PackageImport	–	PackageImport
–	Package	«MetaDataBasketDef»	Package
–	Class	«FunctionDef»	Class
–	Class	«ViewDef»	Class
–	Class	«GraphicDef»	Class
«Property»	Property	–	Property
«Property»	Property	«DrawingRule»	Property
–	Class	«RunTimeParameterDef»	Class
–	PrimitiveType	–	PrimitiveType

correspondence is derived from the fact that the Swiss standard *eCH-0118 GML encoding rules for INTERLIS* specifies an encoding which transforms INTERLIS topics into GML feature collections (Eisenhut, Germann et al. 2011). This transformation is adopted here by mapping the stereotype «TopicDef» to the stereotype «FeatureType» and, to specify that a concrete UML class to which the stereotype «FeatureType» is applied represents a feature collection, by setting its tag definition *isCollection* to *true*. In addition, the UML metaclass *Package* needs to be mapped to the UML metaclass *Class*. This, in turn, requires specifying how those elements which were up to now nested within «TopicDef» UML packages are now represented in the UML model and related to the new «FeatureType» UML class. As «TopicDef» UML packages are mainly used to group UML classes, a suitable solution consists in defining UML associations between the new «FeatureType» UML class and the previously nested UML classes, establishing a relationship in this way similar to the GML feature collection concept.

Also INTERLIS models define feature types in the form of UML classes which, however, are not marked accordingly by means of a suitable stereotype. Only UML classes which represent concepts differing thereof can be provided with corresponding stereotypes. For this reason, all UML classes to which no stereotype is applied are mapped to the stereotype «FeatureType», the tag definition *isCollection* being set to *false* this time.

As regards the concepts *enumeration*, *data type* and *package import*, the mappings provided here assume that these concepts are modelled correctly in INTERLIS application schemas; therefore, on both sides of the table only the corresponding UML metaclasses are listed. A data type, for instance, is used in UML-based INTERLIS models for representing a concept called *structure* which differs from a class in not exhibiting an identity (KOGIS 2006). Also the concept *property* is mapped in this way, with the difference that it is extended by the stereotype «Property» in the Core UML model. The stereotype «DrawingRule» is mapped to the stereotype «Property» as well, since both extend the same UML metaclass.

The stereotype «LineFormTypeDef» can be used to define line structures. According to the INTERLIS reference manual, these line structures always have to extend the INTERLIS structure LineSegment (KOGIS 2006). Since structures are modelled as data types in UML-based INTERLIS models and since the stereotype «LineFormTypeDef» extends the UML metaclass *DataType*, a mapping to the identical UML metaclass in the Core UML profile is specified for this stereotype; the stereotype itself can, however, not be mapped. Furthermore, also units of measure are mapped in the same way. Units of measure are defined in the INTERLIS UML model using the stereotype «UnitDef» which also extends the UML metaclass *DataType*.

Section 5.5.2, page 109, mentions that INTERLIS defines several primitive data types which are modelled in UML-based INTERLIS models using the UML metaclass *PrimitiveType*. Two possibilities for mapping these primitive data types to the Core UML profile exist. One possibility is to map them to the stereotype «Type» from the UML StandardProfileL2 and the corresponding UML metaclass *Class* since also the basic data types defined in the standard ISO/TS 19103 use this representation. The second possibility is to maintain the modelling by mapping the primitive data types to the identical UML metaclass *PrimitiveType* in the Core UML profile. The second option is chosen here. One reason is that the semantics of the concepts type and primitive type are not equivalent. Another reason is provided by the XML-based encoding rule defined in the standard ISO 19118 Annex C. This encoding rules makes use of a stereotype «BasicType», which extends the UML metaclass *Class*, for denoting basic data types and, in particular, also for denoting the basic data types from the standard ISO/TS 19103 (ISO 2011). The encoding rule transforms UML classes marked with the stereotype «BasicType» to XML simple types. The same does the INTERLIS encoding rule mentioned above by transforming the UML primitive types into XML simple types. Since it is not identifiable from the standard ISO 19118, which advantage the use of a UML class has over the use of a UML primitive type, as both concepts are encoded in the same way, the mapping to the UML metaclass *PrimitiveType* is chosen here for the INTERLIS primitive types.

The stereotypes «Leaf», «Union» and «CodeList» from the Core UML profile do not have corresponding concepts in the INTERLIS UML profile. Likewise the stereotypes «MetaDataBasketDef», «FunctionDef», «ViewDef», «GraphicDef» and «RunTimeParameterDef» from the INTERLIS UML profile do not have corresponding concepts in the Core UML profile, which is why only the UML metaclasses extended by these stereotypes are mapped to the Core UML profile.

7.3.4 Implementation of the defined mappings using ATL

The mappings defined above need to be implemented now using a suitable transformation language. Many approaches for transformation languages exist; however, up to now only few standards. In an often-cited survey on transformation languages conducted by (Czarnecki and Helsen 2006), 32 different transformation languages were analysed and grouped according to specific characteristics. To be able to identify the most suited transformation language for executing model transformations

within the context of the above mentioned research project, these 32 transformation languages were evaluated based on the following criteria:

- Does an editor exist which can assist the user in specifying the mappings in the transformation language? If no editor exists, the mappings would have to be defined manually, which can result in a complex task.
- Does a transformation tool exist which can interpret and execute the mappings specified in the transformation language? Transformation languages for which no implementation exists are not considered further as they would require too much development effort in developing usable tools.
- Can the editor and the transformation tool interpret UML models and UML profiles in the form of XMI documents? This aspect is of importance, since the models used in the project are specified using UML and, thus, an editor and transformation tool are required which can interpret these models.
- Are the software implementations based on Eclipse and are they freely available or proprietary? Eclipse-based open source software would be of advantage since Eclipse has already proven in the mdWFS project to be a powerful tool for processing UML models.
- How up-to-date are the software implementations of the editor and the transformation tool?
- Is sufficient documentation and support available for the software implementations and the transformation language itself?
- Is the transformation language user-friendly, the users being employees of the state and federal surveying agencies?

The evaluation revealed that some of these 32 transformation languages are only covered theoretically in scientific publications without applicable implementations being available for them, whereas other transformation languages exhibit implementations which, however, are rather out-of-date as they have not been developed further in recent years; others are based on Triple Graph Grammars (TGG) and, therefore, are rather scientific-oriented and not suitable for the target audience of the project, and still other transformation languages focus on DSLs or embedded systems. In all, the evaluation resulted in three transformation languages appearing most suited for use in the project: QVT Relations, QVT Operational Mappings (cf. section 3.4.3, page 50), and ATL (cf. section 3.4.1, page 43). For all three languages either plug-ins for Eclipse or Eclipse-based implementations exist which are open source, which provide sufficient documentation and support via forums and which were up-to-date at the point in time when the project was conducted.

Since QVT is a specification of the OMG, first the Eclipse-based software mediniQVT (KPIT medini Technologies 2012) and the Eclipse plug-in QVTo (The Eclipse Foundation 2015f) which implement QVT Relations and QVT Operational Mappings, respectively, were employed for specifying the mappings. Both tools were successfully able to transform UML models, however, they provided only a very limited support for transforming UML models to which UML profiles are applied. For this reason, ATL was given a try as third possible language. The Eclipse plug-in available for ATL turned out to be a capable implementation which, in addition, is also able to process UML profiles without any problems. The ATL web site (The Eclipse Foundation 2015a) provides a good documentation and good support through a forum. It lists more than 100 transformation scenarios, amongst others, also for the transformation of MOF-based metamodels and UML profiles. In particular the MDE Case Studies of the Software Languages Lab of the Vrije Universiteit Brussel (Wagelaar 2010) proved to be a very helpful source. All these aspects finally led to the decision, to implement the mappings between the UML profiles using ATL.

Appendix D.1.1, page 193, exemplifies the mappings implemented between the INSPIRE UML profile and the Core UML profile using ATL. The ATL code for mapping the AAA UML profile to the

Core UML profile is equivalent except for the section related to the stereotype «voidable» which is not required in the AAA mapping. For the purposes of demonstrating first and foremost the feasibility of the approach, the tag definitions which were defined as part of the Core UML profile were not considered in these implementations. The ATL transformation definitions were executed successfully by the ATL transformation tool and resulted in a AAA UML model and in a INSPIRE UML model to which the Core UML profile is applied and which do not exhibit a semantic modification of the UML specification any more. Section 7.5.1, page 155, will describe the most important issues regarding the execution of the ATL code.

The mappings between the INTERLIS UML profile and the Core UML profile were not implemented, but it can be expected that they are implementable and executable just as well. The DKM UML model and the DFK UML model do not require transformation definitions to be specified at the metamodel level since to them the Core UML profile is already applied. As mentioned in section 7.2.1, page 140, no UML models existed for these two data specifications in the beginning; therefore, UML models were specifically created which directly are based on the Core UML profile.

7.4 Specification of transformation definitions at the model level using UMLT

This section deals with step three of the workflow, i. e. the specification of transformation definitions at the model layer defining the transformation of the geospatial data based on the source UML models into geospatial data based on the INSPIRE UML model. As a result from the steps one and two in the multi-level transformation workflow (cf. figure 7.1, page 141), the AAA UML model and the INSPIRE UML model are now based on the Core UML profile. This circumstance has to be considered when creating the transformation definitions. Since the project is to reuse developments from the mdWFS project, the transformation language UMLT was employed for defining the transformations.

The complete transformation definition created between the AAA UML model and the INSPIRE UML model (themes CP and AU) is illustrated in figure 7.2. To make the transformation definition universally usable independent of the features actually provided by the input data, all source objects which can theoretically flow into the transformation should be specified as input pins. By doing so, the advantage of the model-driven approach, i. e. that each transformation definition has to be defined only once between specific source and target models, can be employed in the most effective way. However, since the project first and foremost focused on demonstrating the feasibility of the approach, input pins were only defined for those source objects which exist in the geospatial data provided by the surveying agencies. In the same way, output pins were specified only for those target objects which can be created from applying the transformation on the source objects.

As regards the transformation from the AAA UML model to the INSPIRE UML model, four different input pins define the source objects flowing into the structured transformation (*AX_Flurstueck* (cadastral parcel), *AX_KommunalesGebiet* (municipal unit), *AX_Gebietsgrenze* (administrative boundary) and *AX_BesondereFlurstuecksgrenze*) (specific cadastral boundary), whereas five different output pins specify the objects resulting from the transformation process (*CadastralParcel*, *CadastralBoundary* and *CadastralZoning* from the INSPIRE theme CP as well as *AdministrativeUnit* and *AdministrativeBoundary* from the INSPIRE theme AU). Inside the structured transformation, two transformation actions exist which define the individual mappings between the source and target objects. The control flow defines that the transformation tool first executes the transformation action *FSGeometryConverter* and afterwards the transformation action *AAA2INSPIRE*. Furthermore, two

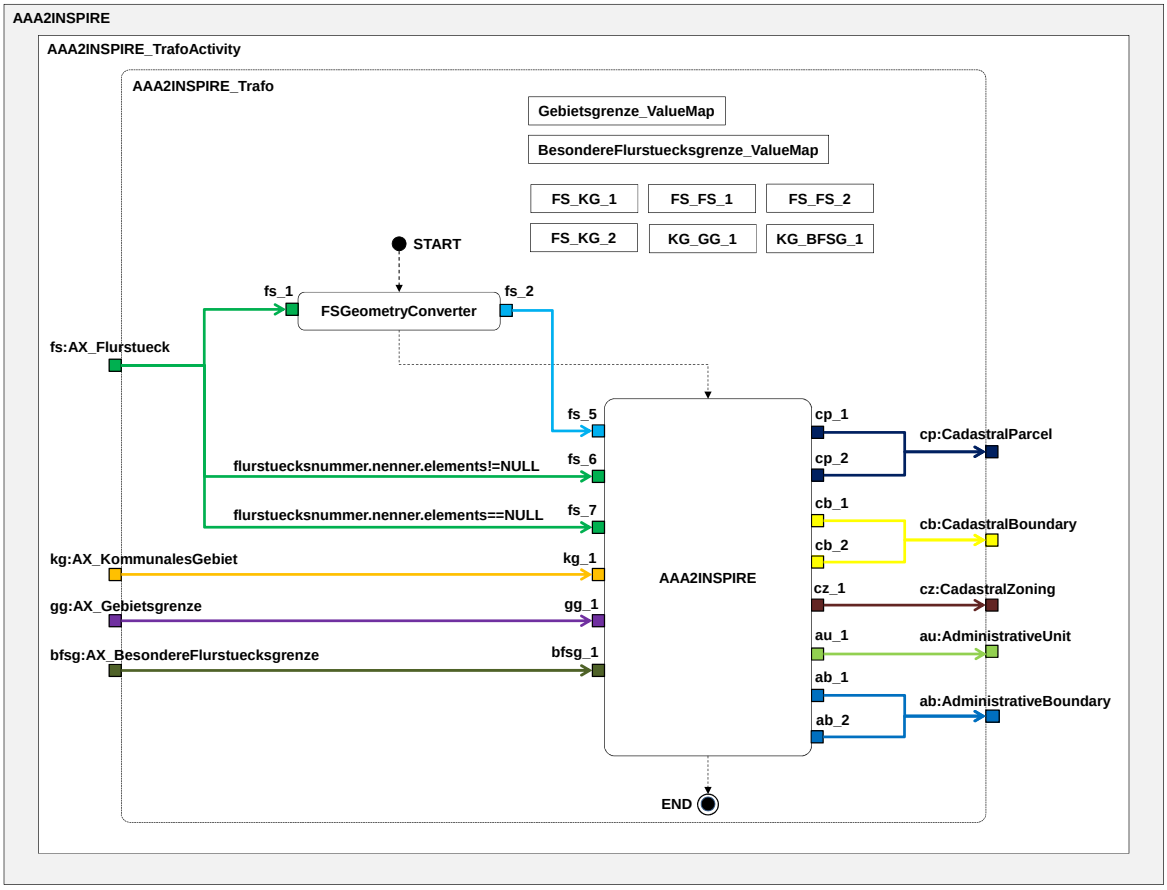


Figure 7.2: UMLT transformation definition between the AAA application schema and the INSPIRE themes Cadastral Parcels and Administrative Units

virtual associations and six value maps are defined within the structured transformation. The detailed definitions of the transformation actions, the transformation value maps and the virtual associations are listed in appendix D.1.2, page 198.

Although the source and target models are now based on the Core UML profile, it was not difficult to create the transformation definition, as was already assumed in section 7.2.1, page 140. The most striking difference results from the use of the new voidable types which are created when mapping the stereotype «voidable» from the INSPIRE UML profile to the Core UML profile. The transformation action *AAA2INSPIRE* specifies, for instance, an assignment definition which assigns the value *Unpopulated* to the attribute *residenceOfAuthority* from the feature type *AdministrativeUnit*. This assignment is represented in the transformation action by the following line:

```
au_1.residenceOfAuthority.voidReason := "Unpopulated"
```

The attribute *residenceOfAuthority* is of the complex type *ResidenceOfAuthority*; thus, without using the Core UML profile, the semantics of UML do not allow for assigning the string value *Unpopulated* to this attribute. However, when mapping the INSPIRE UML profile to the Core UML profile, a specific data type *VoidableResidenceOfAuthority* is created which contains two attributes,

residenceOfAuthority of the type *ResidenceOfAuthority* and *voidReason* of the type *VoidReasonValue*. Since *VoidReasonValue* represents an enumeration and since the value *Unpopulated* is defined as an enumeration literal within this enumeration, *Unpopulated* can now be assigned as value to the attribute *voidReason* without any problems (cf. figure 5.13, page 107, which depicts an equivalent example for the type *DateType* in more detail).

In the same way, transformation definitions were created between the DKM UML model and the INSPIRE UML model (cf. figure 3.15, page 49) as well as between the DFK UML model and the INSPIRE UML model (cf. figure D.7, page 203).

7.5 Execution of the transformations

This section describes, on the one hand, step two of the workflow, i. e. the execution of the transformation definitions specified using ATL, and, on the other hand, the steps five to seven of the workflow, i. e. the actual transformation of the geospatial data based on the transformation definitions specified using UMLT.

7.5.1 Execution of the ATL-based transformation definitions

Before being able to execute the ATL transformation definitions using the ATL transformation engine, the source and target UML models, the corresponding source and target UML profiles and the Core UML profile have to be provided to the engine in the form of separate XMI documents (referred to as XMI model documents and XMI profile documents in the following). As mentioned in section 2.4.4, page 25, the main difficulty in using XMI documents is that each UML tool creates its own XMI dialect, limiting the interoperability of UML models between different UML tools. This problem existed also here. XMI documents generated using EA version 10 were not processable straight away using Eclipse. Therefore, a workaround needed to be devised to obtain the XMI documents based on the Eclipse XMI dialect which consisted in the following three parts:

1. *Retrieving the UML models as XMI model documents*: One possibility to obtain XMI model documents exported from EA version 10 based on the Eclipse XMI dialect is to adjust them manually, which, however, can be a time-consuming and complex task depending on the size of the UML model. Another possibility chosen here was to use the UML tool Magic Draw as auxiliary tool. Magic Draw allows, on the one hand, for importing XMI model documents in different dialects, amongst others, as Enterprise Architect XMI 2.1 documents, and, on the other hand, also for exporting XMI model documents in different dialects, amongst others, as Eclipse UML2 (v3.x) XMI documents. This opportunity was made use of here by importing the INSPIRE and AAA XMI model documents exported beforehand from EA into Magic Draw, exporting them from Magic Draw as Eclipse XMI model documents and importing them finally in Eclipse. These exports and imports worked without any problems, Magic Draw and Eclipse were able to correctly import the XMI model documents without requiring any further manual modifications[4].
2. *Retrieving the UML profiles as XMI profile documents*: The INSPIRE and AdV web sites provide the INSPIRE and AAA UML profiles as XML documents. These XML documents, however,

[4]It is to mention here that after having exported the AAA and INSPIRE UML models from EA as XMI model documents, not the complete UML models were used in the further transformation process. The XMI model documents were manually shortened to only those parts which were required in the context of the INSPIRE themes CP and AU. This also reduced the amount of work necessary for manually applying the stereotypes to the Eclipse XMI model documents in part three of this workaround.

have an EA-specific structure (cf. appendix C.1, page 179) which differs so much from the XMI structure that they cannot be used here. Another possibility to obtain a UML profile as XMI profile document is to export the whole UML model from EA as XMI model document. The UML profile is then represented in the exported XMI model document as part of an additional profile section (cf. listing D.2, page 204) which theoretically simply needs to be stored as XMI profile document of its own (including some further manual adjustments) as Eclipse requires the UML model and the UML profiles as separate XMI documents. However, the export did not work correctly for the INSPIRE and AAA UML profiles since all tag definitions were wrongly represented as stereotypes themselves and not as properties belonging to stereotypes. For this reason, it was chosen to remodel the UML profiles using the UML tool Magic Draw and to export them as Eclipse XMI profile documents. In the same way, also the Core UML profile was modelled using Magic Draw and exported as Eclipse XMI profile document.

3. *Applying the UML profiles to the UML models in Eclipse*: EA XMI model documents are structured differently from Eclipse XMI model documents regarding the application of UML profiles to a UML model as well as the application of stereotypes to UML model elements. To represent the applied UML profiles and stereotypes also in the Eclipse XMI model documents, final adjustments to the structure of the Eclipse XMI model document had to be made manually (cf. appendix D.3, page 204 for a short illustration of this problem).

Afterwards, the ATL transformation engine was successfully able to read the Eclipse XMI documents representing the INSPIRE and AAA UML models as well as the INSPIRE and AAA UML profiles and the Core UML profile and to execute the ATL transformations on them. The results of these transformations are INSPIRE and AAA UML models to which now the Core UML profile is applied.

7.5.2 Execution of the UMLT-based transformation definitions

The transformation of the geospatial data from the source models to the INSPIRE target model is executed based on the transformation definitions specified in the transformation language UMLT using the software FME. As a result from step four of the workflow these transformation definitions exist in the form of XMI documents.

Since the transformation definitions were created between $M_{sourceC}$ and $M_{targetC}$, i. e. between those source models and the INSPIRE target model to which the Core UML profile is applied, the data structures of the source data have to be converted to data structures conforming to $M_{sourceC}$. This is done in step five of the workflow. Due to the fact that the encoding rules for the source models used in the project do not exist in machine-interpretable form, the conversion was solved manually by creating so-called encoding workspaces using FME. The detailed set-up of these encoding workspaces is schematically represented in figure 7.3. For each data format to be transformed to INSPIRE a separate encoding workspace needed to be created. The workspaces read the geospatial data, which are provided in the data formats listed in table 7.2, page 140, by means of suitable FME Readers and write them using the FME FFS Writer to the FME-specific data format FFS (FME Feature Store). To be able to write the geospatial data in the FFS format such that the data conform to $M_{sourceC}$, the FFS Writer needs to know the feature type definitions of $M_{sourceC}$. These feature type definitions are imported and provided to the FFS Writer by means of the XMI Reader from the mdWFS project. The transformations from the various source data formats to the FFS format are not simply one-to-one conversions, but require also taking into account the encoding rules from the source data formats and the XMI Reader encoding. All these issues are considered by the various FME transformers used in the workspace.

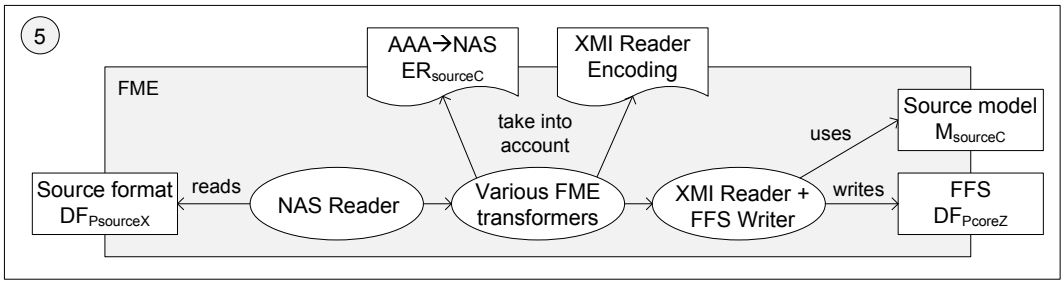


Figure 7.3: Schematic diagram of the encoding workspaces developed in step five of the multi-level information integration workflow for preprocessing the source data such that they conform to the Core UML profile

Next, the actual transformation of the geospatial data from the source models to the target model is conducted, which corresponds to step six of the workflow. The transformations are executed by means of a so-called transformation workspace which is created using FME, too. This transformation workspace needs to be created only once as its definition is so generic that geospatial data conforming to arbitrary data models can be transformed using this workspace. The detailed set-up of the transformation workspace is schematically represented in figure 7.4. The workspace reads the FFS-encoded geospatial data generated in the previous step, executes the UMLT transformation definitions using the UMLTApplier from the mdWFS project and writes the transformed geospatial data again in the FFS format such that the data conform now to $M_{targetC}$.

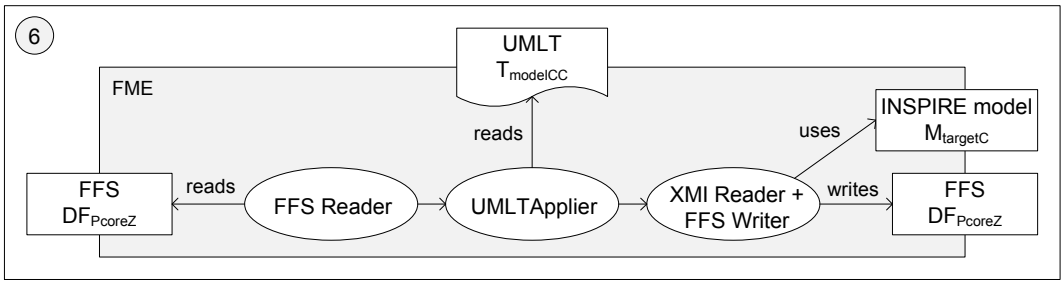


Figure 7.4: Schematic diagram of the transformation workspace developed in step six of the multi-level information integration workflow for transforming the source data to the INSPIRE model

Since the geospatial data conform after their transformation to the data structures of $M_{targetC}$, they finally need to be converted to the data structures conforming to $M_{targetT}$, i. e. the INSPIRE target model to which the INSPIRE UML profile is applied. This is done in step seven of the workflow. Due to the fact that also for the INSPIRE model the encoding rules do not exist in machine-interpretable form, another FME encoding workspace was created. In this step, however, only one encoding workspace is required whose detailed set-up is schematically represented in figure 7.5. The workspace reads the FFS-encoded data generated in the previous step and writes the data to the final INSPIRE GML format. The FME transformers used in the workspace have to take into account the INSPIRE encoding rule and the XMI Reader encoding.

Instead of making use of the FFS format in the steps five to seven, another possibility would have been to stick to the original source and target formats. This would, for instance, mean that in step

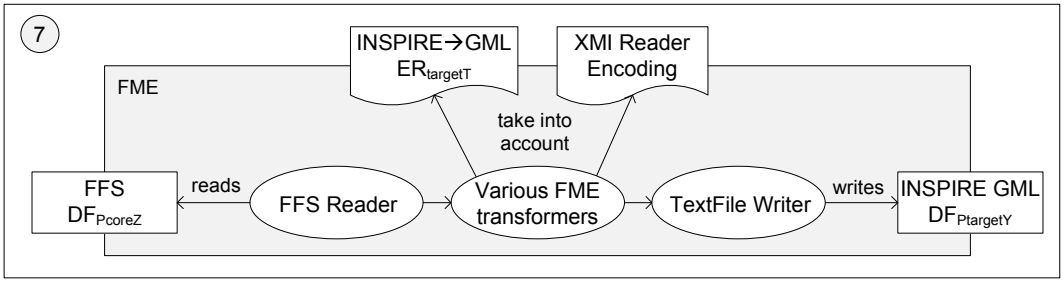


Figure 7.5: Schematic diagram of the encoding workspace developed in step seven of the multi-level information integration workflow for postprocessing the target data such that they conform to the INSPIRE UML profile

five the NAS format would be transformed to the NAS format again, now conforming to the Core UML profile. In step six the NAS format would then be transformed to the INSPIRE GML format conforming to the Core UML profile; and in step seven this format would then be transformed to the final INSPIRE GML format.

The steps five to seven were executed for the data specifications ATKIS/ALKIS, DFK and DKM. By means of the whole workflow implemented for the multi-level information integration, the geospatial data was transformed successfully from the source data models to the INSPIRE themes CP and AU, demonstrating the feasibility of the approach in this way. Furthermore, all INSPIRE GML data sets generated were successfully validated against the INSPIRE GML application schemas.

Figure 7.6 displays the test region around the Lake Constance for which the geospatial data was transformed to INSPIRE. The tables 7.9, 7.10 and 7.11 summarise for each data specification the number of source features flowing into the transformation and the number of target features resulting from the transformation.

Table 7.12 lists the run-times for each data specification. The execution times are shown separately for the encoding workspaces required for preprocessing the source data (step five of the workflow), the actual transformation workspaces (step six of the workflow) and the encoding workspaces required for postprocessing the target data (step seven of the workflow). The transformation was executed using a notebook with Windows 7, SP 1, 64 bit, Intel Core i7-2640M CPU 2.8GHz and 8GB RAM.

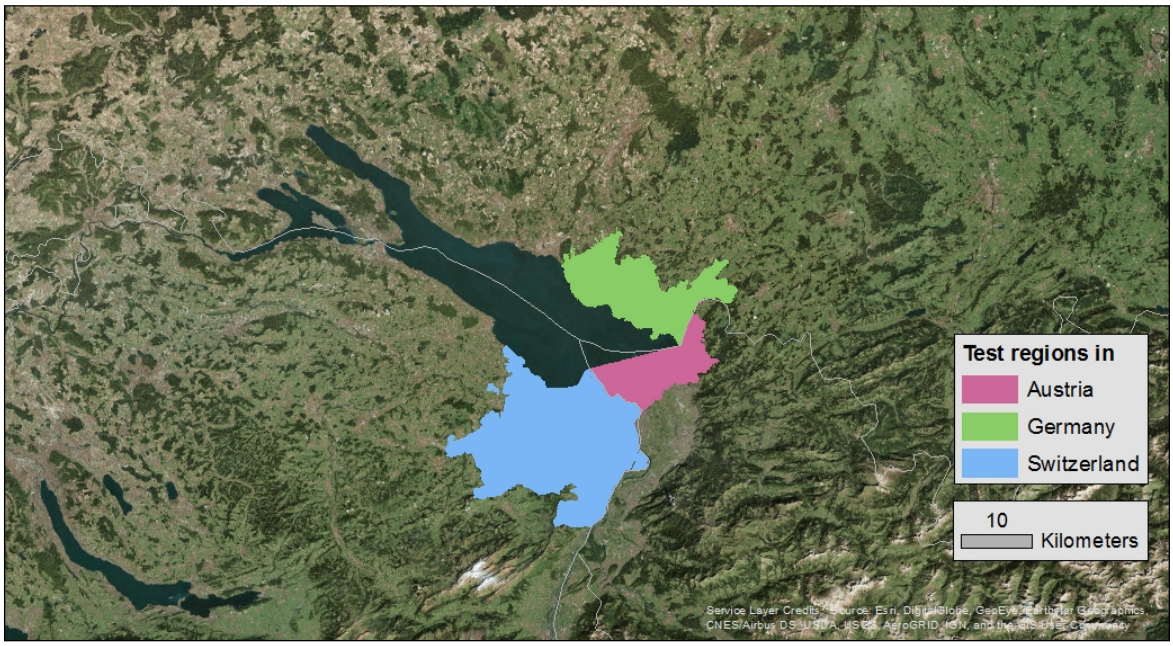


Figure 7.6: The test region around the Lake Constance for which the geospatial data was transformed to INSPIRE

Table 7.9: Overview of the features transformed between the data specification ATKIS/ALKIS and the INSPIRE themes Cadastral Parcels and Administrative Units

ATKIS/ALKIS: Features read		INSPIRE: Features written	
Feature name*	Number of features	Feature name	Number of features
AX_BesondereFlur-stuecksgrenze	3263	CadastralParcel	17056
AX_Flurstueck	17056	CadastralBoundary	68220
AX_Gebietsgrenze	396	CadastralZoning	0
AX_KommunalesGebiet	12	BasicPropertyUnit	0
		AdministrativeUnit	12
		AdministrativeBoundary	396

*AX_BesondereFlurstuecksgrenze (specific cadastral boundary), AX_Flurstueck (cadastral parcel), AX_Gebietsgrenze (administrative boundary), AX_KommunalesGebiet (municipal unit)

Table 7.10: Overview of the features transformed between the data specification DFK and the INSPIRE themes Cadastral Parcels and Administrative Units

DFK: Features read		INSPIRE: Features written	
Feature name*/**	Number of features	Feature name	Number of features
flurstuecke	27623	CadastralParcel	27623
gemarkungen	15	CadastralBoundary	27623
gemeinden	7	CadastralZoning	15
texte	43861	BasicPropertyUnit	0
		AdministrativeUnit	7
		AdministrativeBoundary	7

*flurstuecke (cadastral parcel), gemarkungen (local subdistrict), gemeinden (municipality), texte (text)

**The feature names differ from the input pin names in the UMLT transformation definition in figure D.7, page 203. Since no formal data model was available for the data specification DFK, a data model was specifically defined, trying not to reflect the structure of the source data format one-to-one, but being as platform-independent as possible. Thus, the names and also the quantity of feature types differ. Referring to figure 7.1, page 141, the feature names correspond to $DF_{XsourceS}$, whereas the input pin names correspond to $DF_{XsourceC}$.

Table 7.11: Overview of the features transformed between the data specification DKM and the INSPIRE theme Cadastral Parcels

DKM: Features read		INSPIRE: Features written	
Feature name*	Number of features	Feature name	Number of features
GNR	30162	CadastralParcel	30162
GST	30162	CadastralBoundary	30162
GST_GrundstuecksDB	40460	CadastralZoning	8
		BasicPropertyUnit	30162

*GNR (cadastral parcel number), GST (cadastral parcel), GST_GrundstuecksDB (non-geometric cadastral parcel information)

Table 7.12: Execution times of the transformations

Data specification	Encoding workspace source data	Transformation workspace	Encoding workspace target data
ATKIS/ALKIS	00:10:41	68:00:00*	00:40:00*
DFK	00:00:25	00:13:49	00:02:58
DKM	00:00:41	00:10:56	00:03:23

*These execution times are estimations based on a small subset of the geospatial data transformed. The implementation made use of an sqLite database whose performance was not high enough to process the complete source data at once. The source data needed to be subdivided into small data packages prior to the transformation.

8 Conclusions

This chapter summarises the results achieved within this work, outlines the contributions to scientific research as well as to professional practice and provides an outlook on future research topics evolving from the outcomes of this thesis.

The thesis started with an in-depth introduction of the most fundamental terms and concepts related to geospatial data modelling (chapter 2) and to model-driven transformation of geospatial data (chapter 3). The aim of these two chapters was to establish a coherent terminological basis which is relevant in the geospatial domain in general, but also in the context of this thesis. Whenever suitable, a relationship to relevant standards from the standards organisations ISO, OGC and OMG was established, to emphasise the relevance of the introduced terms and concepts. In particular the ISO 191xx series of geographic information standards plays a major role in geospatial data modelling, which, in turn, relates to several standards from the OGC and the OMG.

Afterwards, the state of the art in geospatial data modelling and model-driven transformation of geospatial data in academia as well as in professional practice was discussed (chapter 4). The first part of this discussion was based on the experience of the author gained from the research projects listed in section 1.1, page 2. This part illustrated, in particular, problems prevalent from the way conceptual models, encoding rules and transformations are currently defined and used in the geospatial domain. The second part of this discussion presented mainly an overview of the various transformation approaches applied in SDI-related projects, commercial and open-source tools as well as in academia and set them in the context of the terms and concepts from the previous two chapters. In addition, this part also introduced the mdWFS project in more detail which constituted the starting point for the research conducted in this thesis.

Next, UML profiles currently in use in the geospatial domain were examined (chapter 5). This examination focused, in particular, on ISO-based UML profiles due to the major role they play in geospatial data modelling. For each UML profile, the stereotypes were discussed as they are currently defined in the UML profile and problems and deficits some of these stereotypes bring about were identified, which, on the one hand, can result in UML profiles not conforming to the UML profile definition of the OMG any more and which, on the other hand, reduce the quality of the UML models to which these UML profiles are applied. Therefore, for each UML profile, a formally defined UML profile was proposed in addition, which eliminates the deficits exposed, illustrating in this way, how the stereotypes can be defined compliant to the UML profile definition of the OMG without loosing the semantics the stereotypes exhibit.

Based on this UML profile examination, solutions for how to cope with UML profiles in geospatial data modelling and in model-driven transformation of geospatial data were presented (chapter 6). This included a general discussion of the findings from the UML profile analysis and the consideration of whether agreeing on one common UML profile or accepting the variety of existing UML profiles would be more advantageous. With the latter option in mind, the examined UML profiles were categorised and, based on this categorisation, a generic concept for developing UML profiles in a structured and reusable way using the UML package merge mechanism was presented which can also be applied to the already existing UML profiles to improve their fitness for use. In addition, a Core UML profile was introduced as a universally applicable, fundamental building block in modelling

and model-driven transformation of geospatial data. The Core UML profile was developed starting with a domain model which specified those concepts considered as fundamental to geospatial data modelling and for which then the semantically most appropriate concepts within the UML metamodel were selected. Furthermore, a multi-level information integration framework was developed which enables the transformation between UML models based on differing and deficient UML profiles. This framework incorporates the Core UML profile as part of a star-converter approach which allows for mapping any source and target UML profile involved in the transformation to the Core UML profile such that the actual model-driven transformation of geospatial data works solely within the framework of the Core UML profile.

Finally, the feasibility and the applicability of the Core UML profile and of the general framework for multi-level information integration was successfully demonstrated by applying it to the transformation of geospatial data from Austria, Germany and Switzerland to the INSPIRE data specifications (chapter 7). This included the definition of transformations at the metamodel layer between the source and target UML profiles and the Core UML profile using the transformation language ATL, the definition of transformations at the model layer between the source and target UML models using the transformation language UMLT as well as the actual transformation of geospatial data from the above mentioned countries based on the software FME.

8.1 Discussion of the results

After this general overview which summarised the individual contributions achieved in each chapter of this thesis, the results of this work will now be discussed based on the research questions stated in section 1.2, page 3.

1. To which extent do the ISO-based UML profiles currently in use in the geospatial domain conform to the UML profile definition of the OMG and in which way do they need to be improved when they exhibit deficits?

   The examination of selected ISO-based UML profiles in use in the geospatial domain revealed that they exhibit several deficits which reduce the quality of those UML models to which they are applied. In the following, the individual deficits which occurred are listed together with solutions for how to improve them.

   - *Semantic correctness*: For some stereotypes, UML metaclasses were chosen because they fit syntactically; however, it was not taken into account whether these UML metaclasses fit semantically as well, to the effect that these stereotypes cause a semantic modification of the UML metamodel, the UML models to which these stereotypes are applied not being interpretable any more in a semantically correct way using standard UML tools. To cope with this deficit, the two-step approach described in (Lagarde et al. 2008; Fuentes-Fernández and Vallecillo-Moreno 2004; Selic 2007) and demonstrated in section 6.3, page 123, can be of help and can lead to improved results. This two-step approach develops first a domain model of those concepts which are to be provided by the UML profile and maps these domain concepts afterwards to semantically suitable UML metaclasses. In this way, the risk of focusing too much on syntactical aspects from the very start of the UML profile design process can be reduced. This approach is also applied in some of the related works from academia (cf. section 4.3.4, page 70) and in individual UML profiles developed by the OMG. This approach should in particularly be made use of when new UML profiles are to be developed or when existing UML profiles are

to be revised; this applies also to the current revision of the standards ISO/TS 19103 and ISO 19109, which in their current state rather introduce new deficits to the UML profiles defined therein (cf. sections 5.1.3, page 85, and 5.2, page 85). As regards other existing UML profiles, the communities which define these UML profiles should work towards finding solutions for their stereotypes in question, e. g. by revising their UML profiles using the two-step approach as well. In cases, where no satisfying solution can or is intended to be found, it needs to be considered whether the advantage of this stereotype outweighs the quality loss this stereotype imposes on the UML model or whether the stereotype maybe should not be used at all.

- *UML 1 vs. UML 2*: All evaluated UML profiles were defined against the background of UML 1, although most UML models to which they are applied are modelled using UML 2. UML 1 allowed for adding tag definitions to UML model elements without the need of defining them as constituent of a corresponding stereotype beforehand. Similarly, also stereotypes were allowed to be applied to UML model elements without the existence of a corresponding UML profile. UML 2 does not allow for these practices any more. Thus, tag definitions and stereotypes which conform to UML 1, but which are applied to UML models conforming to UML 2, turn these UML models into UML models which do not conform to UML 2 any more. A further problem observed was that in UML 2, some UML metaclasses from which to extend a stereotype have changed. For instance, to assign a stereotype to an attribute or an association in UML 1, the stereotype needed to extend the UML metaclasses *Attribute* and *AssociationEnd*, whereas in UML 2 the stereotype needs to extend now the UML metaclass *Property*. These problems can be solved by defining all UML profiles explicitly anew based on UML 2 and by strictly following the UML profile definition of the OMG in the UML profile design process. This is, in fact, generally advisable since, as the revision documents ISO/DIS 19103 and ISO/DIS 19109 indicate, a shift from UML 1 to UML 2 will take place in the ISO 191xx series of geographic information standards in the near future, to the effect that all communities which base their UML models on these standards will have to switch to UML 2 sooner or later as well. Also, UML models which are defined using UML 1 and which are to be transformed to UML 2, need to take into account these differences in their transformation definition.
- *Formal definition of UML profiles*: All evaluated UML profiles are only defined in tabular and/or textual form, the definitions sometimes being distributed in various parts of the particular specification document, which makes it difficult to get a clear picture of the concepts actually provided by the UML profile. This deficit can be solved by defining the UML profile formally according to the UML profile definition of the OMG and by providing in the specification document at least a UML profile diagram which allows for implementing the UML profile unambiguously by any person which needs to make use of the UML profile. Actually, it is not necessarily required that the UML profile is defined in a formal way. The UML profile rather needs to be described such precisely using natural language that it can be implemented unambiguously. Otherwise, there is the risk that the UML profile is interpreted and implemented in different ways by different persons. Unambiguously defined UML profiles, in contrast, ensure that they are used, without exception, in every UML tool in exactly the same way and based on the same semantics.
- *UML keywords*: The evaluated UML profiles do not differentiate between UML stereotypes and UML keywords. Although UML keywords are visually equivalent to UML stereotypes, they represent notation elements which are part of the UML syntax. UML keywords serve in distinguishing between metamodel elements which use the same visual notation, they do not adapt the UML metamodel to specific platforms or domains. Within the evaluated UML

profiles, however, UML keywords were used equivalent to UML stereotypes; even wrong UML metaclasses were denoted for them in some cases. This deficit can be solved by simply making use of those UML model elements, when defining UML models, to which the corresponding UML keywords belong and by removing these UML keywords from the UML profile. For some of these UML keywords, tag definitions were specified in the UML profile. In this case, the corresponding UML metaclass needs to be extended by a stereotype and the tag definitions need to be defined as properties of this stereotype. Furthermore, the evaluated UML profiles also make use of stereotypes which are already predefined by the UML specification as part of the UML profile StandardProfileL2. When tag definitions need to be added to such a predefined stereotype, this stereotype should be specialised by an equally named stereotype which contains the desired tag definitions. This is possible without any problems since the stereotypes from the StandardProfileL2 belong to a different UML package and, thus, to a different namespace than the stereotypes of the UML profile to be defined.

2. How must a formally correct and universally applicable UML profile for geospatial data modelling be designed and which core concepts must it contain?

   The answer to the first part of this question can actually be derived from the answer to the previous question.
   - A formally correct and universally applicable UML profile must conform to the UML 2 profile definition of the OMG, which means, it must not cause any semantic modification of the UML metamodel, all required tag definitions must be defined as properties of corresponding UML stereotypes and the stereotypes must extend UML metaclasses from the UML 2 metamodel. Furthermore, the UML profile design process preferably makes use of the two-step domain model approach.

   To be universally applicable and, in particular, also to promote a widespread use of geospatial data models and geospatial data itself beyond the geospatial domain, the UML profile needs to be as slender as possible, as the discussion in section 6.3.1, page 124, showed, which leads to the following answer for the second part of the question.
   - Within this thesis, a Core UML profile was developed which contains those concepts considered as essential for geospatial data modelling. Since the ISO 191xx series of geographic information standards plays a major role in geospatial data modelling, these concepts were first of all derived from the ISO/TS 19103 UML profile and the ISO 19109 UML profile based on the finding that the concepts defined therein are relevant to the entire geospatial domain and, thus, should be represented in the Core UML profile in any case. These concepts are, for the most part, common to all evaluated UML profiles, which means, the Core UML profile basically represents an intersection of all concepts specified within the existing UML profiles. The Core UML profile was, in addition, complemented by further concepts considered as essential.
   - Furthermore, this Core UML profile specifies only concepts which are relevant at the conceptual level, i. e. concepts which are required for creating platform-independent models. To keep the model clean from any superfluous content, platform-specific aspects are, in accordance with the MDA approach, not considered as beneficial at this stage of the modelling process, but should only be added to the UML model during a PIM→PSM transformation.
   - The usability of the Core UML profile was demonstrated twofold. On the one hand, it served as an intermediate step in a multi-level information integration framework, assisting there in mapping differing and deficient community UML profiles to a common and semantically correct

core based on which it was possible to perform the model-driven transformation of geospatial data without problems. On the other hand, this Core UML profile proved to be very useful in reverse engineering conceptual models for data specifications which are not yet defined formally, the Core UML profile guaranteeing that the most fundamental and relevant concepts of the geospatial domain are automatically at hand for this task.

3. How can the variety of UML profiles existing in the geospatial domain be structured and designed in a modular way?

   The UML profile analysis showed that, even if a specific UML profile makes use of concepts which are already defined as stereotypes within another UML profile, this specific UML profile needs to define the stereotypes anew to be able to use them in the development of UML models, to the effect that new deficits may arise from these continual redefinitions. Furthermore, the analysis revealed that the scope of the concepts differs. While some concepts focus on the representation of general conceptual aspects in UML models, others serve in adding community-specific aspects to UML models and still others focus on the provision of encoding aspects. Based on these findings, this research question can be answered as follows.

   - A classification was introduced which allows for structuring the various UML profiles according to base UML profiles and community conceptual UML profiles, which only contain conceptually-specific aspects, as well as according to general encoding UML profiles and community encoding UML profiles, which only contain encoding-specific aspects. This classification is based on the fact that all evaluated UML profiles build on the same base UML profiles and on the same general encoding UML profiles, i. e., they reuse the concepts defined therein and extend them merely by additional concepts required within a certain community. This classification requires those UML profiles which currently define conceptual as well as encoding-specific aspects within the same UML profile to be split up into two separate UML profiles, which, however, does not present a problem to the UML profiles evaluated in this thesis. The classification, furthermore, was designed such that general encoding UML profiles are completely community-independent and, thus, can be reused by any community encoding UML profile wishing to transform its platform-independent UML model to a specific platform.
   - The classification was formally specified by making use of UML package merge which allows for combining the contents of individual UML profiles in a modular way. Within this modular specification, the base UML profiles represent the merged packages whose contents are integrated into the community UML profiles, which represent the receiving packages. This modular concept can be applied to all UML profiles already existing in the geospatial domain and, in particular, also to UML profiles which are still to be created, evading in this way from the beginning the necessity of having to define every stereotype anew. Furthermore, this concept also withstands the evolution of existing UML profiles.

4. Is it possible to apply the approach of model-driven transformation of geospatial data in spite of the variety of UML profiles in use, in particular, when these UML profiles do not conform to the UML profile definition of the OMG?

   This question is based on the idea introduced in (Kutzner and Eisenhut 2010) that limitations in meta-interoperability between models to which differing and deficient UML profiles are applied can be solved by applying a multi-level information integration process which makes use of a common UML profile as an intermediate step. It leads to the following answer.

- Model-driven transformation of geospatial data can be conducted when reducing the various UML profiles to a common denominator, i. e. to those concepts defined by the common UML profile which, in addition, conforms to the UML profile definition of the OMG. For this reason, a framework was developed which works within the limits of such a common UML profile. The common UML profile used in the framework is the Core UML profile as it represents exactly those concepts which are at the core of the geospatial domain. The framework follows the MDA approach, i. e. it operates at three layers, the metamodel layer, the model layer and the instance layer. At the metamodel layer, the differing UML profiles are mapped to the Core UML profile such that all source and target UML models are now based on the Core UML profile and, thus, exhibit the same common denominator and the same model quality, eliminating the limitations in meta-interoperability in this way. Furthermore, also the source geospatial data and the source and target encoding rules need to be converted such that they conform to the Core UML profile. Afterwards, the general concept for model-driven transformation of geospatial data can be applied in the usual way.
- The feasibility and the applicability of the framework was successfully demonstrated. First, it was possible to create transformation definitions between several community UML profiles and the Core UML profile and to execute these transformations using the transformation language ATL. Second, it was possible to create transformation definitions between source and target UML models to which the Core UML profile is applied using UMLT. And third, it was possible to automatically transform the source geospatial data based on these UMLT transformations using the software FME.
- Due to the slenderness of the Core UML profile, only those concepts from the community UML profiles can be mapped to the Core UML profile which are either core concepts themselves or which can be mapped to one of the core concepts due to a related semantics. This means, concepts from a source UML profile which cannot be mapped to the Core UML profile can also not be propagated further to a target UML profile. This, however, is not seen as disadvantageous as it can be assumed that such concepts exist in one specific UML profile only. For concepts existing in several UML profiles it should rather be considered whether they might be fundamental to the whole geospatial domain and, thus, should become part of the Core UML profile.

8.2 Contributions of the results to scientific research and professional practice

The results of this thesis can contribute to scientific research and to professional practice in many ways. First of all, the framework brings about the following general advantages:

- Since the framework requires object-oriented UML models for the source and target data, the user can also create the UMLT transformation definitions based on an object-oriented view, which simplifies this definition process. For instance, no transformations need to be defined for relationships (joins, foreign keys, relationship tables), which, in contrast, would be required when defining the transformations based on a Relational view.
- The clear graphical representation of the UMLT transformation definitions allows for a very good traceability of the individual transformation rules and is, thus, ideally suited for communication purposes between domain experts, comparable to UML class diagrams as regards data models. A limitation to the usability when creating the transformation definitions based on UML models to which the Core UML profile is applied was hardly to identify. In particular the ISO-based AAA and

INSPIRE models change only to a very small extent from a user's point of view. Models which are not ISO-based might experience a stronger transformation; the mappings defined for INTERLIS in section 7.3.3, page 149, however, showed a very good coverage of concepts from which can be implied that the modification is also here manageable.

- Further well-known advantages of the general model-driven transformation approach are transferable to the multi-level information integration framework as well. These are, for instance, format-independence since the UMLT transformation definitions are valid independent of a specific data format (provided that the format can be derived from the UML model using encoding rules) and reusability since the UMLT transformation definitions are valid independent of a certain transformation tool.

Nevertheless, certain prerequisites need to be fulfilled to be able to apply the multi-level information integration framework. These prerequisites are listed in table 8.1.

Table 8.1: Prerequisites required to be able to apply the multi-level information integration framework

Prerequisites	Degree of fulfilment
• The geospatial data need to be described by machine-interpretable conceptual UML models.	Partially fulfilled. UML models still mainly represent models for communication purposes. Manual adjustments may be required.
• UML models to which semantically wrong stereotypes are applied need to be transformed into semantically correct UML models.	Fulfilled. The transformations can be executed using ATL.
• Data specifications which are not yet defined conceptually need to be reverse-engineered and need to apply a suitable UML profile.	Fulfilled. The reverse-engineering process can, in addition, directly make use of the Core UML profile.
• The encoding rules need to be machine-interpretable.	Not fulfilled. Currently only ambiguously, in natural language defined encoding rules exist.
• The community UML profiles need to be defined formally and to be available either as UML profile diagram or as XMI document.	Not fulfilled. Currently only EA-specific documents exist.
• The user needs to have knowledge about	
– ATL	Fulfillable. But only required when the mapping between the community UML profile and the Core UML profile does not yet exist.
– UMLT	Fulfillable.
– the source and target UML models	Fulfillable, in case the UML models are publicly accessible.
• Specific software tools are required	
– FME as well as the FME extensions XMI Reader and UMLTApplier	Fulfilled.
– UMLT editor	Fulfilled.
– Eclipse incl. EMF, ATL and UMLT plugins	Fulfilled.

Furthermore, the discussion about UML profiles can contribute to scientific research and to professional practice in the following way:

- *Contributions of chapter 2 and 3*: The introduced terms and concepts are encountered regularly in research and daily practice, however, often are not used in a coherent way. Also, some of these terms originate from the computer science domain, but are used differently in the geospatial domain. In these cases, a relationship to their original meaning was established to put them in the correct context again. These chapters are to serve as fundament to any interested person intending to get familiar with geospatial data modelling and model-driven transformation of geospatial data.
- *Contributions of chapter 4*: This chapter illustrates predominant problems from the way conceptual models are currently defined and used in the geospatial domain, which were encountered by the author. The aim thereby is to make modellers aware of similar pitfalls and to point out how important an adequate knowledge of the UML specification is.
- *Contributions of chapter 5 and 6*:
  - The proposed formal ISO UML profiles as well as the development of the Core UML profile using the two-step domain model approach are to demonstrate how UML profiles should preferably be developed.
  - The discussion in these chapters is to make aware of the fact that the definition of formal UML profiles does not only require sufficient knowledge of the UML profile mechanism, but also of the UML metamodel itself. Otherwise, erroneous UML profiles are the result.
  - In addition, it is to be pointed out that, although current UML tools such as Enterprise Architect allow for adding tag definitions to UML models without specifying a stereotype for them, this approach is discouraged from, in particular, when at the same time the requirement states that the UML models have to be defined compliant to UML 2.
  - The use of UML profiles can be facilitated by providing a formal UML profile diagram and by providing a working XMI document (validity of the XMI document is difficult to achieve) which can be imported by UML tools (or at least by those tools prevalent in the geospatial domain, since interoperability for XMI documents is still limited).
  - When new UML profiles need to be defined, or when existing UML profiles need to be extended by new stereotypes and/or tag definitions, it has to be taken care that these new concepts are defined such that no semantic modification of the UML metamodel occurs and that the stereotypes extend those UML metaclasses belonging to the correct UML version.
  - Last, the discussion is to point out that the existence of a plurality of UML profiles can result in a multitude of complexities and it needs to be considered whether this is in fact wanted.

8.3 Future research topics

From the discussions and outcomes of this thesis the following possible topics for future research can be derived:

- The UML profile examination and the solutions provided in this thesis are based on a selection of UML profiles. In practice, more UML profiles exist such as the Dutch NEN3610 UML profile. It needs to be analysed whether the solutions (in particular the generic concept for developing UML profiles and the multi-level information integration framework) also withstand other UML profiles. This is of particular interest regarding non-ISO-based UML profiles.
- The generic concept for developing UML profiles makes use of the UML package merge concept. It needs to be tested whether current UML tools are able to implement the generic concept as intended

or whether specific tools would need to be developed. Furthermore, it needs to be tested how the UML package merge usability is when UML profiles resulting from a merge are applied to UML models, in particular when PIM models are to be transformed to PSM models which then require the application of encoding-specific aspects from the encoding UML profiles.

- Section 6.3.1, page 124, states that the domain model exhibits similarities with the GFM from the standard ISO 19109. It would be useful to analyse these similarities in more detail. Furthermore, since the GFM represents a domain model itself, it would make sense to map the concepts from the GFM to a UML profile of its own and to compare this GFM UML profile with the Core UML profile. A similar statement as regards the GFM was already formulated in (Einspanier 2005): 'For better enabling language integration and reuse of models, as in the case of the defined GFM view on UML models, a clear and unambiguous profile would be required. This could e.g. be achieved by providing more specific stereotypes that properly identify the corresponding GFM constructs.'
- Section 3.2.1, page 33, describes different types of heterogeneity which can exist in information integration. This categorisation refers to information in general, aspects inherent to geographic information are not considered, except that heterogeneity can occur with respect to the geometric representation of spatial objects as raster or vector data. Here an examination of its own would be required which takes into account also aspects such as different levels of detail and different modelling paradigms for representing geometries.
- This thesis strongly focuses on the OMG/MDA technical space. It should be investigated, in which way the transition, called projection (Bézivin 2006), to other technical spaces, in particular to the RDF and XML technical spaces, can be conducted, for instance, against the background of the model-driven transformation since also other technical spaces allow for transforming data. Similarly, (Kutzner and Eisenhut 2010) state that the use of XSLT for transforming GML application schemas is convenient to a limited extent only since XSLT is targeted at the XML paradigm, whereas GML complies to the OO paradigm, the same applies to RIF as it corresponds to the RDF paradigm. This topic also relates to the expressive power of a language. Not all transformation definitions might be expressible to the same extent using different transformation languages and also data models may vary in their ability of representing the same universe of discourse. Also, the encoding of geospatial data as defined in the standard ISO 19118 (cf. section 3.3.1, page 36) might be influenced by the expressive power of the source system, the target system and the data transfer format. A more detailed examination is required to which extent modelling and transformation languages which are based on different paradigms, belong to different technical spaces and exhibit a different expressive power can be used jointly.
- Owing to the non-existence of formally defined and automatically executable encoding rules, the encoding was solved manually in the prototypical implementation of the multi-level information integration framework by defining FME encoding workspaces. It should be analysed, whether suitable formal languages for the definition and automatic execution of the encoding rules exist; if this is not the case, a suitable language needs to be created. Furthermore, to promote the MDA approach in the geospatial domain and to establish a wide-spread use of automatic encoding, encoding rules for those data formats most common in the geospatial domain need to be defined using the chosen language and made publicly available.
- The multi-level information integration framework makes use of the transformation language UMLT. This language was already developed in 2006, when model transformation was still in its early stages and not many other, already applicable transformation languages existed. Therefore, it would be necessary to conduct a thorough investigation of existing transformation languages taking into account UMLT as well. Furthermore, the database and data warehouse domains should

be included in this examination as well since these domains deal with the integration of data based on heterogeneous data models already since the 1970s (Kutzner and Eisenhut 2010).

- The UMLT transformation definitions are currently defined using the UMLT editor and are then mapped onto the internal transformation functionalities of FME. It should be investigated, whether the FME workbench could directly be used to model and execute the UMLT transformation definitions. This will probably require extending FME by conceptual schema awareness beyond the current possibility of FME readers and writers to take into account data models.

A Metamodels

A.1 The General Feature Model

Figure A.1 shows an extract from the General Feature Model specified in the standard ISO 19109 which represents the concepts defined for features.

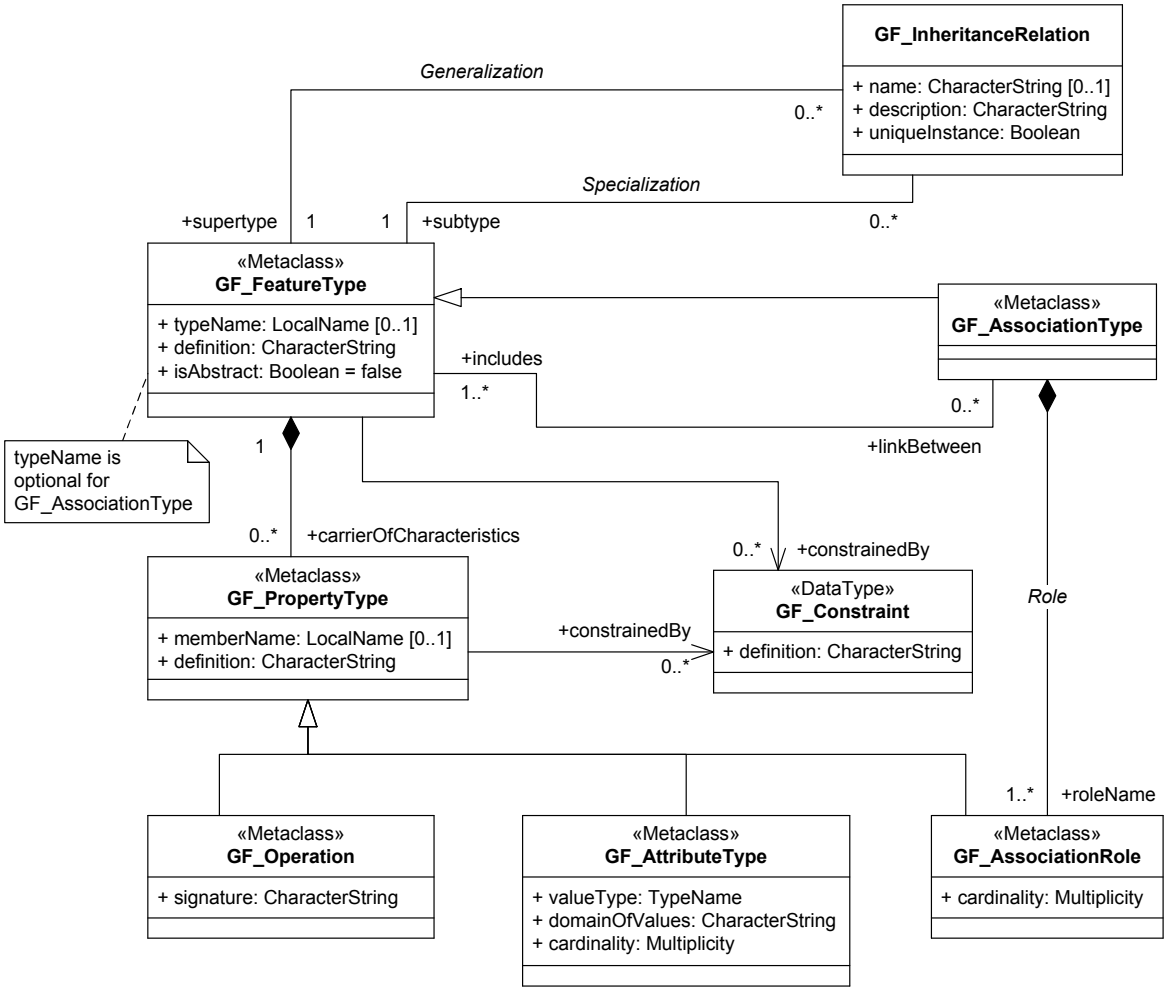


Figure A.1: Extract from the General Feature Model (ISO 2005a)

A.2 UML 2 Superstructure – Classes package

The *Classes* package of the UML Superstructure defines the metamodel elements required for modelling UML class diagrams. The package reuses packages defined in the UML Infrastructure, in particular the *Kernel* package which provides 'the core modeling concepts of the UML, including classes, associations, and packages' (ISO 2012b) and extends them 'with additional features, associations, or superclasses' (ISO 2012b).

In the following, two diagrams of the *Kernel* package are depicted, the *Classes* diagram in figure A.2 and the *DataTypes* diagram in figure A.3. The diagrams are included, because they clearly show that the UML metamodel elements *Class* and *DataType* are direct subclasses of the UML metamodel element *Classifier* and that the UML metamodel elements *PrimitiveType* and *Enumeration*, in turn, are subclasses of *DataType*.

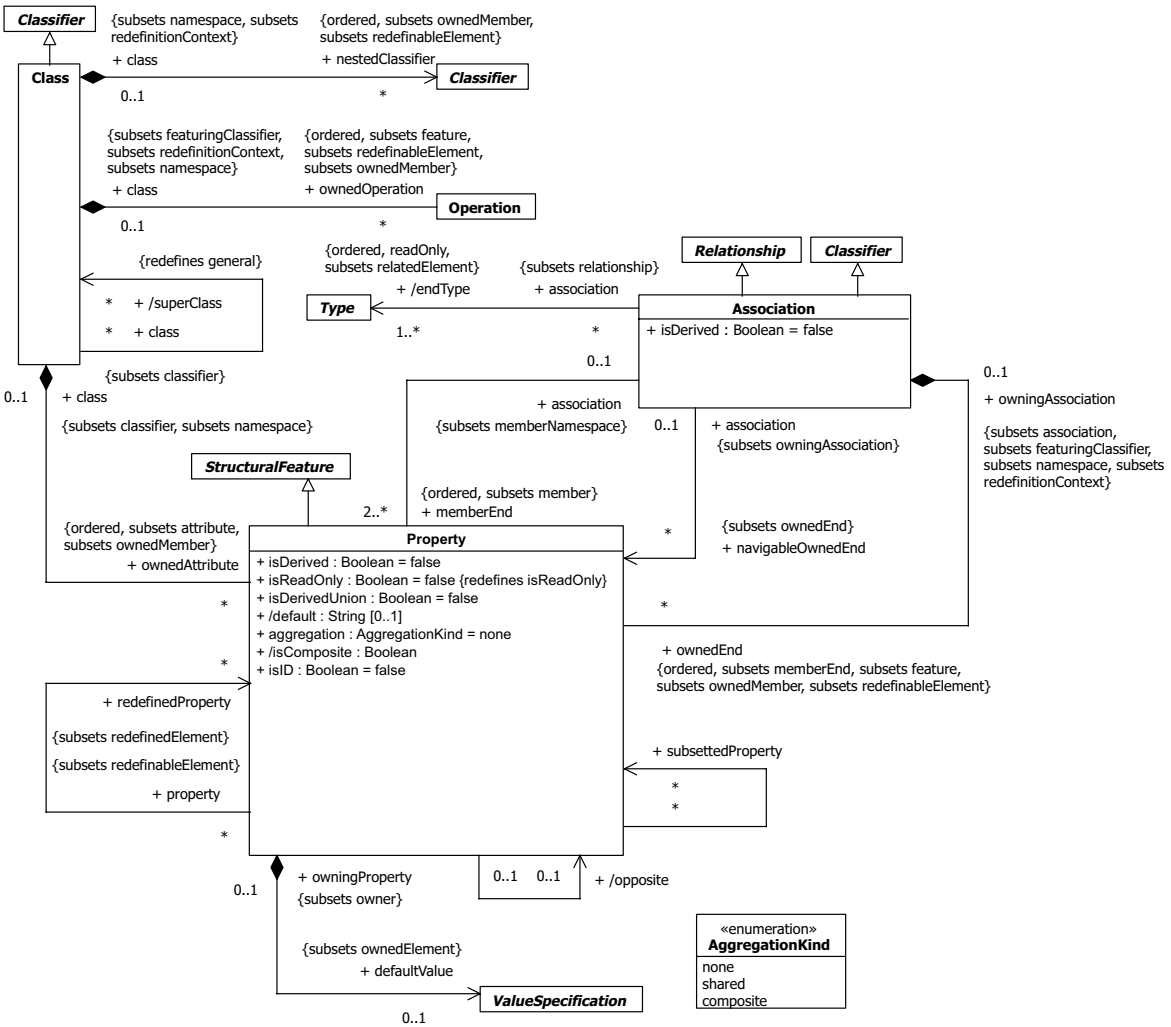


Figure A.2: *Classes* diagram of the Kernel package (ISO 2012b)

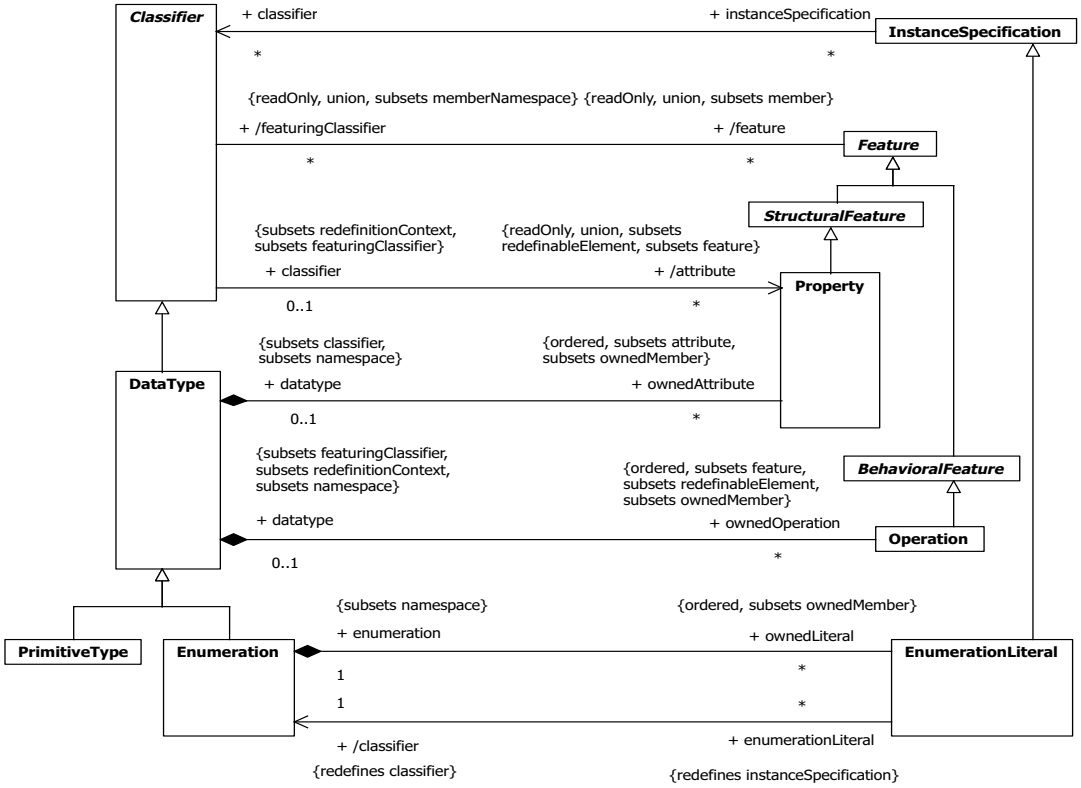


Figure A.3: *DataTypes* diagram of the Kernel package (ISO 2012b)

B Encoding of geospatial data

B.1 Concept of data interchange between two systems according to the standard ISO 19118

Figure B.1 displays the concept of data interchange between two systems which is specified in the standard ISO 19118 (cf. section 3.3.1, page 36).

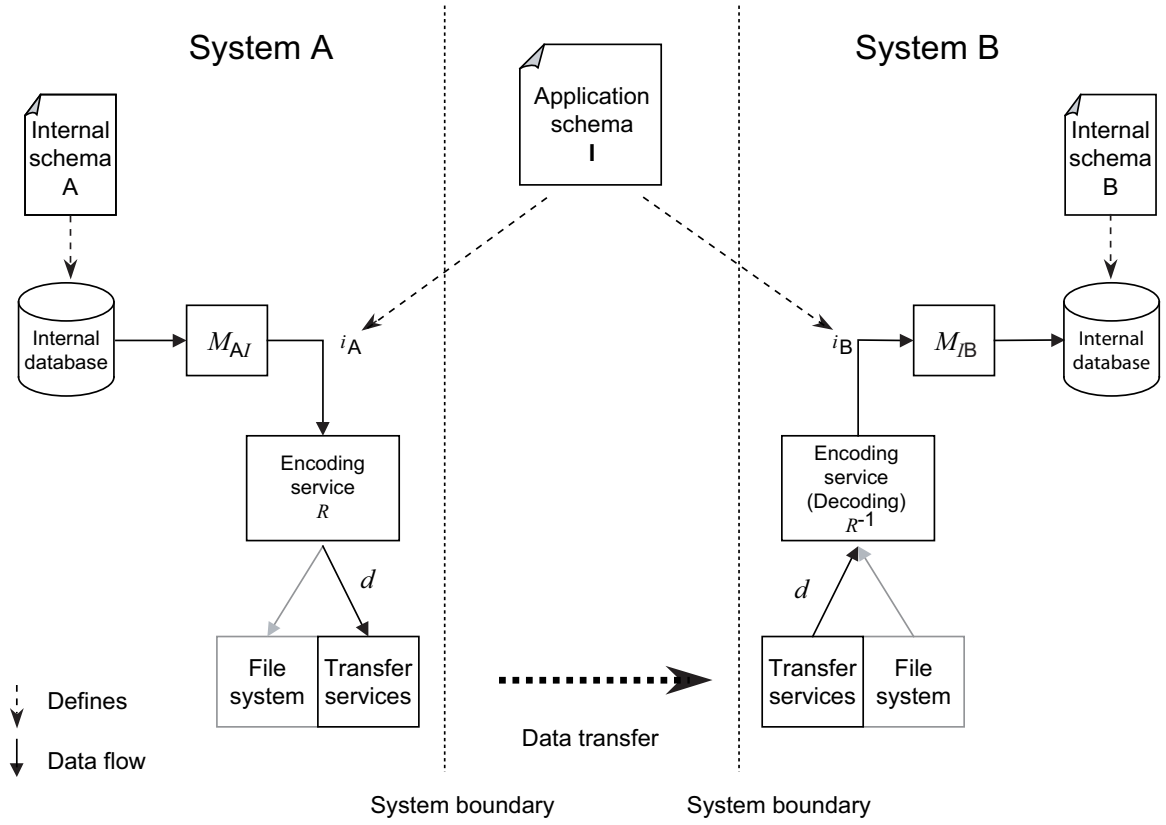


Figure B.1: ISO 19118 concept of data interchange between two systems (ISO 2011)

B.2 GML encoding of spatial attributes

The following listings illustrate the different GML encodings of spatial attributes modelled according to either value semantics or reference semantics (cf. section 4.2.3.2, page 62):

- Listing B.1 represents two building objects as INSPIRE GML instances which correspond to the UML model in figure 4.5, page 63. The spatial attribute *geometrySolid* is defined in the UML model using *value semantics*.
- Listing B.2 displays the same two building objects as CityGML instances which correspond to the UML model in figure 4.6, page 63. Here, the spatial attribute *lod3Solid* is defined using *reference semantics*.

Listing B.1: INSPIRE GML encoding of the spatial attribute *geometrySolid* modelled using value semantics

```
<bu-core3d:Building gml:id="building1">
  <bu-core3d:geometry3DLoD3>
    <bu-core3d:BuildingGeometry3DLoD3>
      <bu-core3d:geometrySolid>
        <gml:Solid gml:id="solid1">
          <gml:exterior>
            <gml:Shell>
              <gml:surfaceMember>
                <gml:CompositeSurface gml:id="compsurface1">
                  <gml:surfaceMember>
                    <gml:Polygon gml:id="wallSurface4711">
                      <gml:exterior>
                        <gml:LinearRing>
                          <gml:posList>...</gml:posList>
                        </gml:LinearRing>
                      </gml:exterior>
                    </gml:Polygon>
                  </gml:surfaceMember>
                  ...
                </gml:CompositeSurface>
              </gml:surfaceMember>
            </gml:Shell>
          </gml:exterior>
        </gml:Solid>
      </bu-core3d:geometrySolid>
    </bu-core3d:BuildingGeometry3DLoD3>
  </bu-core3d:geometry3DLoD3>
</bu-core3d:Building>

<bu-core3d:Building gml:id="building2">
  <bu-core3d:geometry3DLoD3>
    <bu-core3d:BuildingGeometry3DLoD3>
      <bu-core3d:geometrySolid>
        <gml:Solid gml:id="solid2">
          <gml:exterior>
            <gml:Shell>
              <gml:surfaceMember>
                <gml:CompositeSurface gml:id="compsurface2">
                  <gml:surfaceMember>
                    <gml:Polygon gml:id="wallSurface0815">
                      <gml:exterior>
                        <gml:LinearRing>
                          <gml:posList>...</gml:posList>
                        </gml:LinearRing>
                      </gml:exterior>
                    </gml:Polygon>
```

```
                    </gml:surfaceMember>
                    ...
                  </gml:CompositeSurface>
                </gml:surfaceMember>
              </gml:Shell>
            </gml:exterior>
          </gml:Solid>
        </bu-core3d:geometrySolid>
      </bu-core3d:BuildingGeometry3DLoD3>
    </bu-core3d:geometry3DLoD3>
</bu-core3d:Building>
```

Listing B.2: CityGML encoding of the spatial attribute *lod3Solid* modelled using reference semantics (Gröger et al. 2012, modified)

```
<bldg:Building gml:id="building1">
  <bldg:lod3Solid>
    <gml:Solid gml:id="solid1">
      <gml:exterior>
        <gml:CompositeSurface gml:id="compsurface1">
          <gml:surfaceMember>
            <gml:Polygon gml:id="wallSurface4711">
              <gml:exterior>
                <gml:LinearRing>
                  <gml:posList>...</gml:posList>
                </gml:LinearRing>
              </gml:exterior>
            </gml:Polygon>
          </gml:surfaceMember>
          ...
        </gml:CompositeSurface>
      </gml:exterior>
    </gml:Solid>
  </bldg:lod3Solid>
</bldg:Building>

<bldg:Building gml:id="building2">
  <bldg:lod3Solid>
    <gml:Solid gml:id="solid2">
      <gml:exterior>
        <gml:CompositeSurface gml:id="compsurface2">
          <gml:surfaceMember>
            <gml:OrientableSurface orientation="-">
              <gml:baseSurface xlink:href="#wallSurface4711"/>
            </gml:OrientableSurface>
          </gml:surfaceMember>
          ...
        </gml:CompositeSurface>
      </gml:exterior>
    </gml:Solid>
  </bldg:lod3Solid>
</bldg:Building>
```

C UML profiles

C.1 Publicly available UML profiles for Enterprise Architect

A UML profile, called *UML Profile for GML Applications Schemas*, is publicly available from the Solid Earth and Environment Grid community web site (SEE Grid 2009). The UML profile is based on the encoding rule specified in the standard ISO 19136 Annex E (cf. section 5.3, page 86). It can be used within Enterprise Architect (EA) for modelling UML application schemas which are to be encoded as GML application schemas. The UML profile is provided as XML document in an EA-specific format; listing C.1 shows the UML profile.

Based on the XML document, a formal UML profile diagram was created for this thesis to gain a better overview of the stereotypes and tag definitions the UML profile contains. The UML profile diagram is displayed in figure C.1. The tag definitions in the XML document do not define any types and multiplicities, thus, suitable types and multiplicities were added to the UML profile diagram based on the descriptions given for each tag definition. The types *Integer*, *String* and *Boolean* are primitive types which are predefined in the UML 2 Infrastructure and which can be used in defining MOF-based metamodels (ISO 2012a). Several tag definitions in the XML document specify sets of predefined values from which to choose when the corresponding stereotypes are applied to model elements. These values were defined in the UML profile diagram as enumerations based on the values provided in the XML document. Some tag definitions also provide default values which were adopted in accordance with the XML document, except for the stereotypes «class» and «bundle». For them, the XML document provides a default value for the tag definition *xsdEncodingRule* which differs from all other stereotypes and, thus, was corrected to be consistent with the other stereotypes.

Another UML profile, the *UML Profile for INSPIRE data specifications*, is publicly available from the INSPIRE web site (JRC 2012). This UML profile is based on the INSPIRE UML profile specified in the INSPIRE documents D2.5 and D2.7 (cf. section 5.4.1, page 101) and can be used within EA for modelling INSPIRE application schemas. Figure C.2 shows the formal UML profile diagram which was created for this UML profile based on the available XML document. The modelling decisions described above were applied here in exactly the same way.

Listing C.1: XML document of the *UML Profile for GML Applications Schemas* for Enterprise Architect (SEE Grid 2009). The XML attributes *description* and *notes* were omitted to confine the size of the listing.

```
<?xml version="1.0" encoding="UTF-8"?>
<UMLProfile>
  <!-- modified by Simon Cox from profile created by Clemens Portele - last edit: 2009-05-152 -->
  <Documentation id="GML" name="UML Profile for GML Applications Schemas" version="2.0"/>
  <Content>
    <Stereotypes>

      <Stereotype name="Application Schema">
        <AppliesTo>
          <Apply type="package"/>
        </AppliesTo>
        <TaggedValues>
          <Tag name="targetNamespace" default="FIXME"/>
          <Tag name="xmlns" default="FIXME"/>
```

```
            <Tag name="version" default="FIXME"/>
            <Tag name="xsdDocument" default="FIXME"/>
            <Tag name="gmlProfileSchema"/>
            <Tag name="xsdEncodingRule" values="iso19136_2007 | iso19139_2007 |
                  iso19136_2007_INSPIRE_Extensions" default="iso19136_2007"/>
          </TaggedValues>
        </Stereotype>

<!-- this stereotype may be added for packages without stereotype during the conversion to an implementation model, if
       needed
        <Stereotype name="bundle">
          <AppliesTo>
            <Apply type="package"/>
          </AppliesTo>
          <TaggedValues>
            <Tag name="xsdDocument"/>
            <Tag name="xsdEncodingRule" values="iso19136_2007 | iso19139_2007 |
                  iso19136_2007_INSPIRE_Extensions" default="iso19136_2007_INSPIRE_Extensions" />
          </TaggedValues>
        </Stereotype> -->

        <Stereotype name="Leaf">
          <AppliesTo>
            <Apply type="package"/>
          </AppliesTo>
          <TaggedValues>
            <Tag name="xsdDocument"/>
            <Tag name="xsdEncodingRule" values="iso19136_2007 | iso19139_2007 |
                  iso19136_2007_INSPIRE_Extensions" default="iso19136_2007"/>
          </TaggedValues>
        </Stereotype>

        <Stereotype name="FeatureType">
          <AppliesTo>
            <Apply type="class"/>
          </AppliesTo>
          <TaggedValues>
            <Tag name="xsdEncodingRule" values="iso19136_2007 | iso19139_2007 |
                  iso19136_2007_INSPIRE_Extensions" default="iso19136_2007"/>
            <Tag name="noPropertyType" values="false" default="false"/>
            <Tag name="byValuePropertyType" values="false" default="false"/>
            <Tag name="isCollection" values="true | false" default="false"/>
          </TaggedValues>
        </Stereotype>

        <Stereotype name="Type">
          <AppliesTo>
            <Apply type="class"/>
          </AppliesTo>
          <TaggedValues>
            <Tag name="xsdEncodingRule" values="iso19136_2007 | iso19139_2007 |
                  iso19136_2007_INSPIRE_Extensions" default="iso19136_2007"/>
            <Tag name="noPropertyType" values="false" default="false"/>
            <Tag name="byValuePropertyType" values="false" default="false"/>
            <Tag name="isCollection" values="true | false" default="false"/>
            <Tag name="xmlSchemaType"/>
          </TaggedValues>
        </Stereotype>

<!-- this stereotype may be added for classes with no stereotype during the conversion to an implementation model
        <Stereotype name="class">
          <AppliesTo>
            <Apply type="class"/>
          </AppliesTo>
          <TaggedValues>
```

```
            <Tag name="xsdEncodingRule" values="iso19136_2007 | iso19139_2007 |
                  iso19136_2007_INSPIRE_Extensions" default="iso19136_2007_INSPIRE_Extensions" />
            <Tag name="noPropertyType" values="false" default="false" />
            <Tag name="byValuePropertyType" values="false" default="false" />
            <Tag name="isCollection" values="true | false" default="false" />
          </TaggedValues>
        </Stereotype> -->

        <Stereotype name="DataType">
          <AppliesTo>
            <Apply type="class"/>
          </AppliesTo>
          <TaggedValues>
            <Tag name="xsdEncodingRule" values="iso19136_2007 | iso19139_2007 |
                  iso19136_2007_INSPIRE_Extensions" default="iso19136_2007"/>
            <Tag name="noPropertyType" values="false" default="false"/>
            <Tag name="isCollection" values="true | false" default="false"/>
          </TaggedValues>
        </Stereotype>

        <Stereotype name="Union">
          <AppliesTo>
            <Apply type="class"/>
          </AppliesTo>
          <TaggedValues>
            <Tag name="xsdEncodingRule" values="iso19136_2007 | iso19139_2007 |
                  iso19136_2007_INSPIRE_Extensions" default="iso19136_2007"/>
            <Tag name="noPropertyType" values="false" default="false"/>
          </TaggedValues>
        </Stereotype>

        <Stereotype name="Enumeration">
          <AppliesTo>
            <Apply type="class"/>
          </AppliesTo>
          <TaggedValues>
            <Tag name="xsdEncodingRule" values="iso19136_2007 | iso19139_2007 |
                  iso19136_2007_INSPIRE_Extensions" default="iso19136_2007"/>
          </TaggedValues>
        </Stereotype>

        <Stereotype name="CodeList">
          <AppliesTo>
            <Apply type="class"/>
          </AppliesTo>
          <TaggedValues>
            <Tag name="xsdEncodingRule" values="iso19136_2007 | iso19139_2007 |
                  iso19136_2007_INSPIRE_Extensions" default="iso19136_2007"/>
            <Tag name="asDictionary" values="true" default="true"/>
            <Tag name="codeSpace" type="string" values="" default=""/>
            <Tag name="dictionaryIdentifier" type="string" values="" default=""/>
            <Tag name="memberIdentifierStem" type="string" values="" default=""/>
          </TaggedValues>
        </Stereotype>

        <Stereotype name="Import">
          <AppliesTo>
            <Apply type="dependency"/>
          </AppliesTo>
        </Stereotype>

<!-- this stereotype may be added for properties with no stereotype during the conversion to an implementation model -->
        <Stereotype name="property">
          <AppliesTo>
            <Apply type="attribute"/>
```

```
          <Apply type="associationRole"/>
        </AppliesTo>
        <TaggedValues>
          <Tag name="sequenceNumber"/>
          <Tag name="inlineOrByReference" values="inline | byReference | inlineOrByReference"
               default="inlineOrByReference"/>
          <Tag name="isMetadata" values="true | false" default="false"/>
        </TaggedValues>
      </Stereotype>

    </Stereotypes>
  </Content>
</UMLProfile>
```

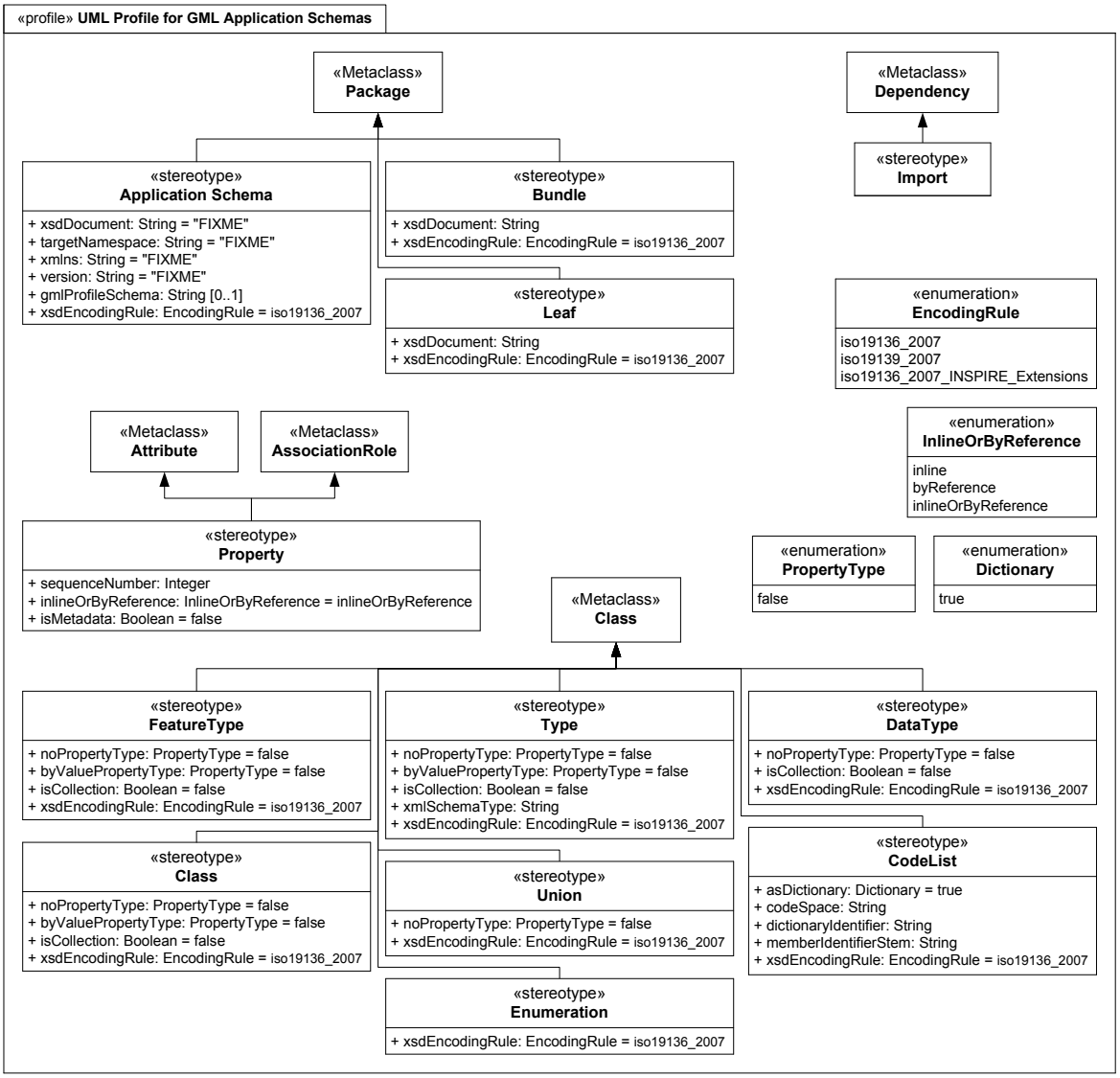


Figure C.1: Formal UML profile diagram created based on the EA-specific *UML Profile for GML Application Schemas*

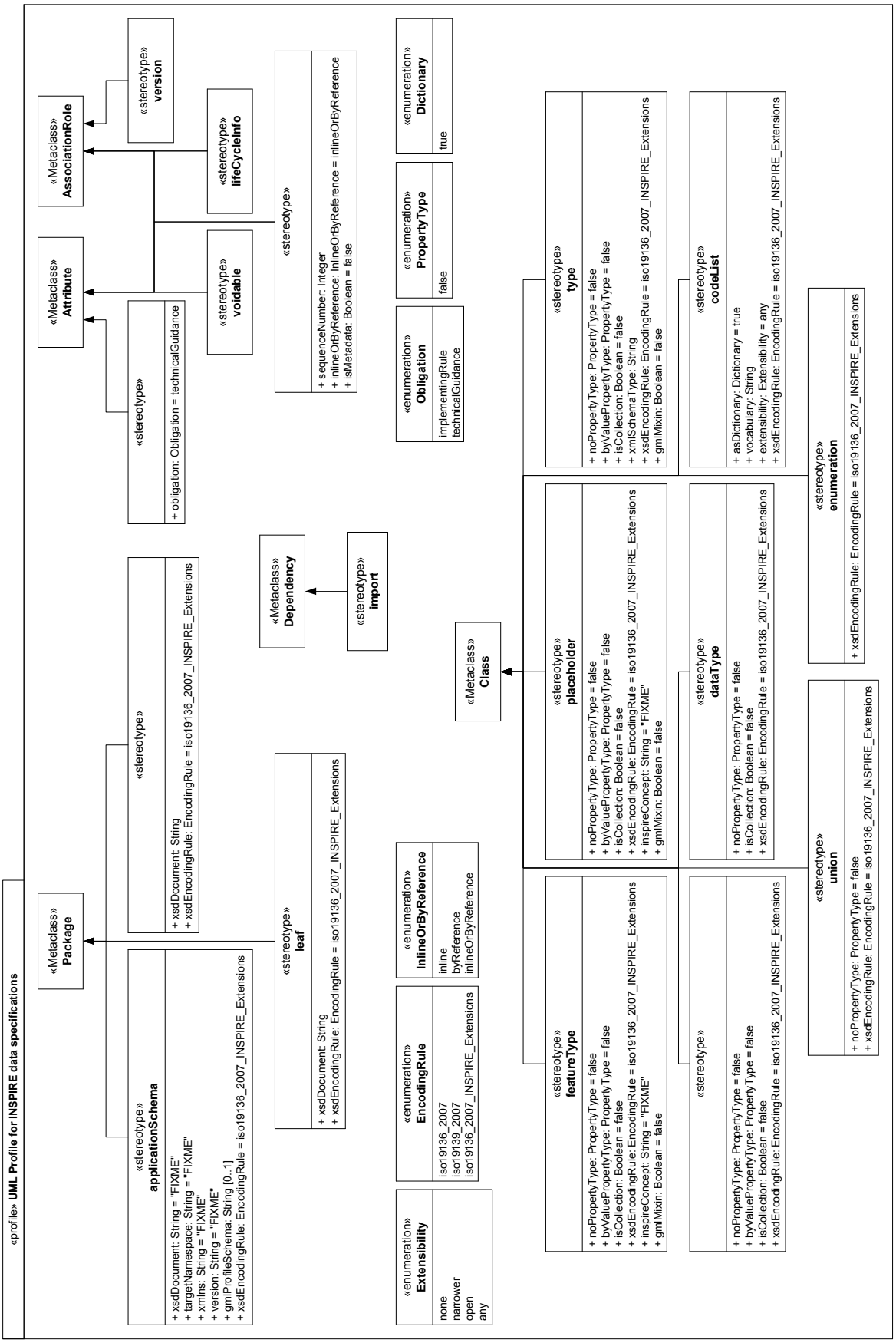


Figure C.2: Formal UML profile diagram created based on the EA-specific *UML Profile for INSPIRE data specifications*

C.2 UML profile diagrams of ISO standards and ISO draft international standards

Figure C.3 displays the formal UML profile diagram which is provided in the document ISO/DIS 19103 (cf. section 5.1.3, page 85).

In contrast, also the document ISO/DIS 19109 defines a UML profile, but only in tabular form together with further definitions within the running text. Therefore, a formal UML profile diagram was created for this thesis based on the information provided in (ISO 2013b) (cf. section 5.2, page 85). The diagram is shown in figure C.4. The tag definitions in the DIS do not define types and multiplicities, thus, suitable types and multiplicities were added to the formal UML profile based on the descriptions given for each tag definition. The type *String* is a primitive type predefined in UML 2.

Similarly, also the ISO 19136 UML profile is only defined in tabular and textual form by the standard ISO 19136 Annex E (cf. section 5.3, page 86). Figure C.5 displays the formal UML profile diagram which was created based on the information provided in this standard.

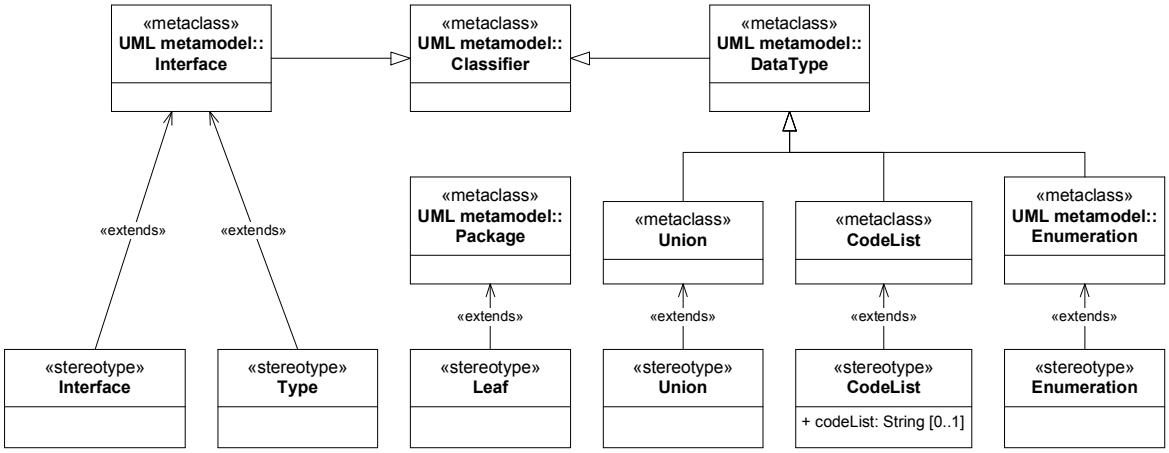


Figure C.3: Formal UML profile diagram provided in ISO/DIS 19103 (ISO 2013a)

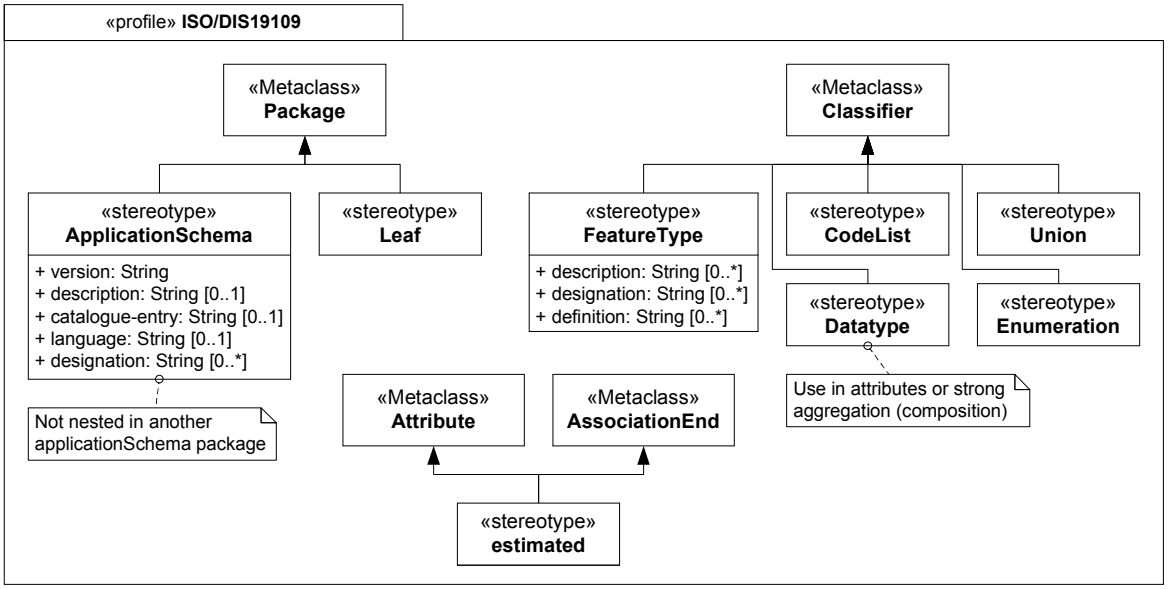


Figure C.4: Formal UML profile diagram created based on the descriptions in ISO/DIS 19109

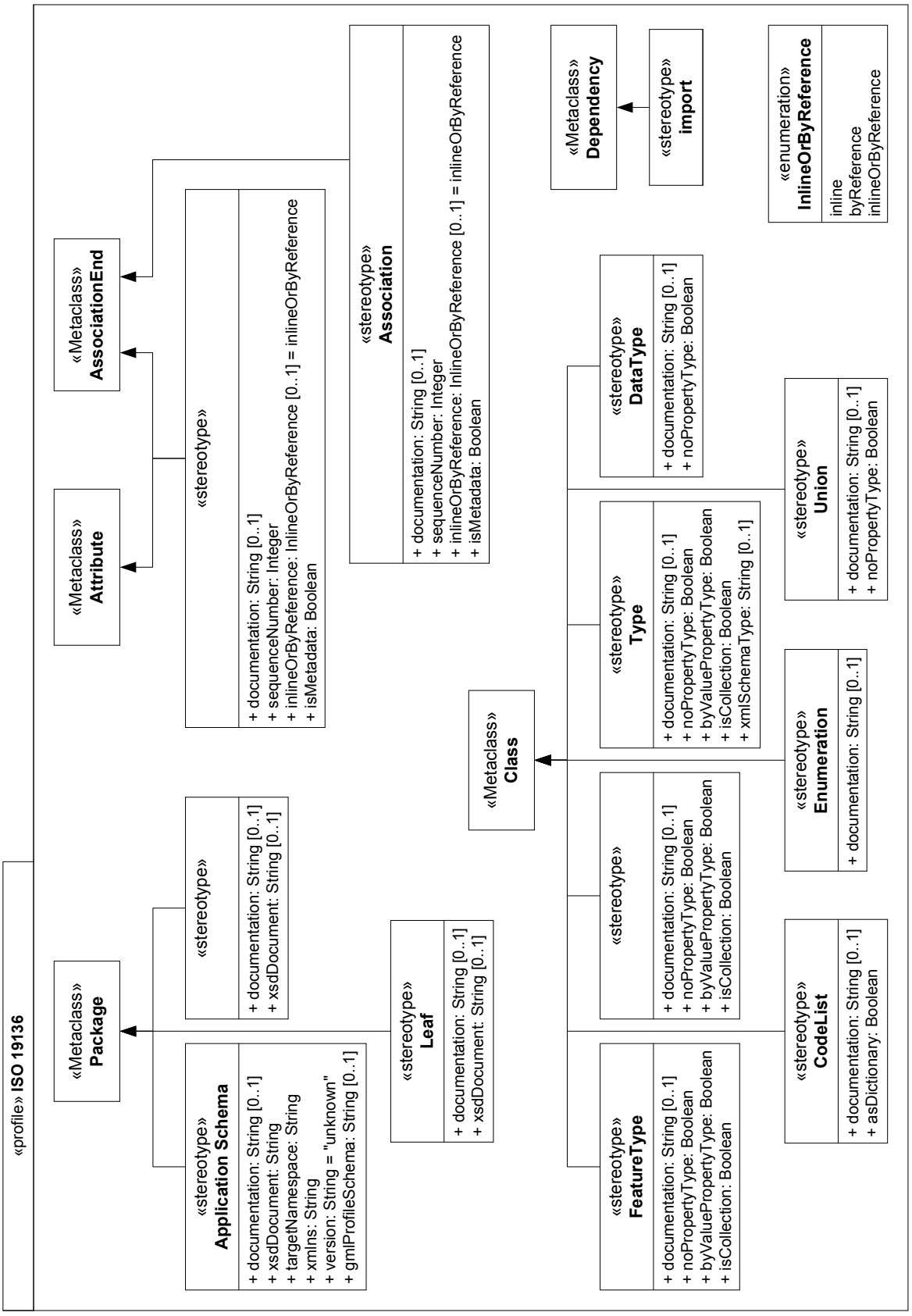


Figure C.5: Formal UML profile diagram created based on the descriptions in ISO 19136 Annex E

C.3 INSPIRE UML profile

As mentioned in section 5.4, page 100, the INSPIRE UML profile is based on the standard ISO 19136 Annex E and, thus, reuses the stereotypes from there, complementing them by additional tag definitions. In the following, these new tag definitions defined for each of the stereotypes will be provided, together with information regarding their meaning, types, multiplicities and intended use, which is deduced from the descriptions given in the INSPIRE documents D2.5 and D2.7.

The stereotypes «applicationSchema», «leaf», «union», «enumeration» and «dataType», UML packages with any or no stereotype as well as UML classes representing object types specify the new tag definition *xsdEncodingRule* which is listed in table C.1. Furthermore, each of the stereotypes «featureType», «type» and «codeList» is provided with several new tag definitions which are listed in the tables C.2, C.3 and C.4, respectively. Above that, the following issues were noticed in the documents D2.5 and D2.7:

- As regards the tag definition *gmlMixin*, the INSPIRE documents contain contradictory information. On the one hand, the document D2.7 states that '[m]ixin classes shall have either the stereotype <<featureType>> or no stereotype and the tagged value "gmlMixin" with a value "true"' (JRC 2014b), on the other hand, the document D2.5 lists this tag definition also for the stereotype «type».
- All stereotypes which are reused from the ISO 19136 UML profile theoretically must also contain the tag definition *documentation* since this tag definition was defined there already. However, the INSPIRE UML profile does not make use of this tag definition and no information is provided, whether this tag definition was omitted on purpose.
- The INSPIRE document D2.5 lists the tag definition *isCollection* for the stereotype «dataType» which, however, does not make sense. The tag definition was originally defined in ISO 19136 Annex E, but not for data types; also, no corresponding conversion rule is defined in the INSPIRE document D2.7. In addition, the description provided for this tag definition gives the appearance that it is listed mistakenly.

Similarly to the other UML profiles, also the INSPIRE UML profile is only specified in tabular form together with further definitions in the running text of the INSPIRE documents D2.5 and D2.7 (cf. section 5.4.1, page 101). Therefore, the formal UML profile diagram displayed in figure C.6 was created for this thesis based on the information provided in these documents.

In addition, figure C.7 shows the UML profile diagram of the formal INSPIRE UML profile which is proposed in section 5.4.2, page 104. This UML profile makes use of UML package merge.

Table C.1: INSPIRE UML profile: Additional tag definition of the stereotypes «applicationSchema», «leaf», «union», «enumeration» and «dataType», of UML packages with any or no stereotype and of UML classes representing object types

Name	Meaning	Type	Multiplicity	Use
xsdEncodingRule	The encoding rule to be applied	String	0..1	GML encoding

Table C.2: INSPIRE UML profile: Additional tag definitions of the stereotype «featureType»

Name	Meaning	Type	Multiplicity	Use
inspireConcept	URI reference to the feature concept	String	1	Conceptual level, GML encoding
gmlMixin	The feature type is a mixin class	Boolean	1	GML encoding
xsdEncodingRule	The encoding rule to be applied	String	0..1	GML encoding

Table C.3: INSPIRE UML profile: Additional tag definitions of the stereotype «type»

Name	Meaning	Type	Multiplicity	Use
gmlMixin	The type is a mixin class	Boolean	1	GML encoding
xsdEncodingRule	The encoding rule to be applied	String	0..1	GML encoding

Table C.4: INSPIRE UML profile: Additional tag definitions of the stereotype «codeList»

Name	Meaning	Type	Multiplicity	Use
extensibility	the code list is extensible by a third party	String	1	Conceptual level, GML encoding
vocabulary	URI of the code list registry	String	0..1	Conceptual level, GML encoding
xsdEncodingRule	The encoding rule to be applied	String	0..1	GML encoding

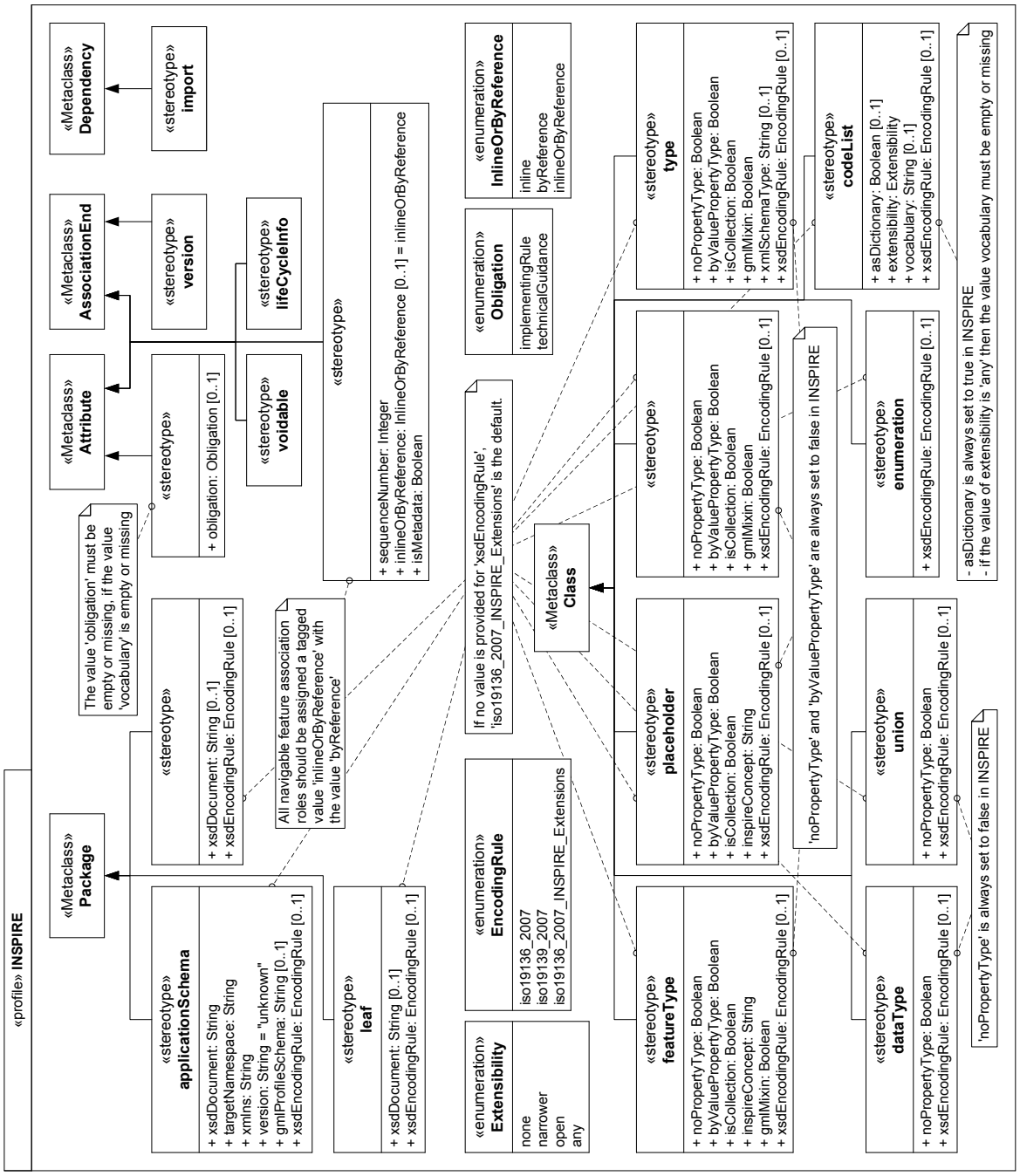


Figure C.6: Formal UML profile diagram created based on the descriptions in the INSPIRE documents D2.5 and D2.7

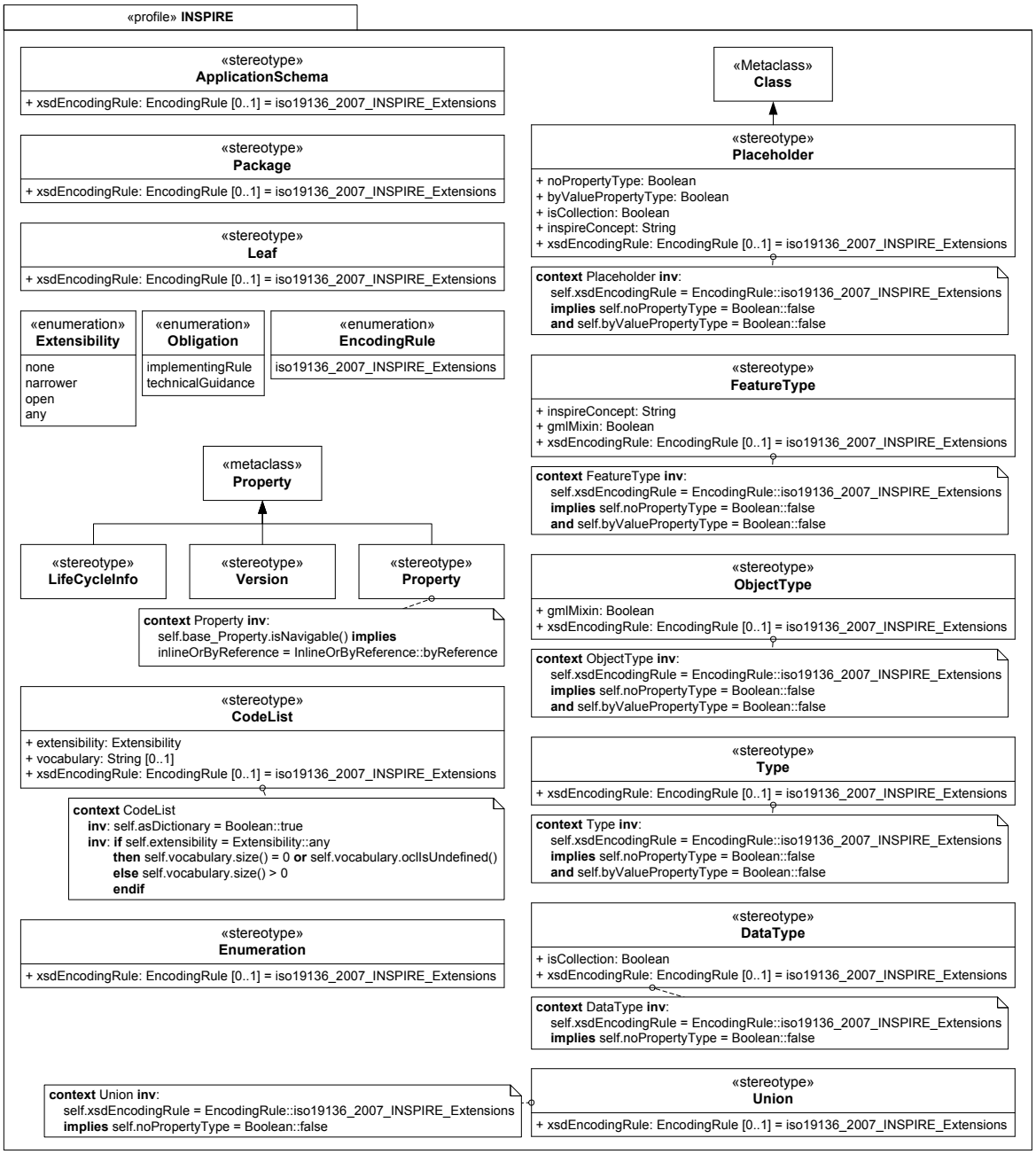


Figure C.7: Proposed formal INSPIRE UML profile

C.4 Core UML profile

Listing C.2 shows the XMI structure of the Core UML profile which was developed in section 6.3, page 123. The Core UML profile was implemented using Eclipse UML2 version 3.2.1. The XMI document contains Ecore-specific information which is required by Eclipse to be able to process the UML profile. This information is not shown here since it can automatically be added again within Eclipse. The XMI code shown in the listing can also be imported by Magic Draw without any problems.

Listing C.2: Eclipse XMI document of the Core UML profile

```
<?xml version="1.0" encoding="UTF-8"?>
<uml:Profile xmi:version="2.1" xmlns:xmi="http://schema.omg.org/spec/XMI/2.1"
       xmlns:ecore="http://www.eclipse.org/emf/2002/Ecore" xmlns:uml="http://www.eclipse.org/uml2/3.0.0/UML"
       xmi:id="id1" name="CoreUMLProfile" metamodelReference="id1">
  <eAnnotations xmi:id="_XEoTsMhGEeG82fPhPvPIyg" source="http://www.eclipse.org/uml2/2.0.0/UML">
    <!-- ... Ecore-specific content ... -->
  </eAnnotations>
  <packageImport xmi:id="id1">
    <importedPackage xmi:type="uml:Model" href="pathmap://UML_METAMODELS/UML.metamodel.uml#_0"/>
  </packageImport>
  <packageImport xmi:id="id2">
    <importedPackage xmi:type="uml:Model" href="pathmap://UML_LIBRARIES/UMLPrimitiveTypes.library.uml#_0"/>
  </packageImport>
  <packagedElement xmi:type="uml:Stereotype" xmi:id="s1" name="ApplicationSchema">
    <ownedRule xmi:id="a1" name="NoNestedApplicationSchemas" constrainedElement="s1">
      <specification xmi:type="uml:OpaqueExpression" xmi:id="sp1">
        <language>OCL2.0</language>
        <body>inv: self.base_Package.nestedPackage->closure(nestedPackage)->forAll(c | c.extension->forAll(p | not
              p.ownedEnd.type.name='ApplicationSchema'))</body>
      </specification>
    </ownedRule>
    <ownedAttribute xmi:id="a2" name="base_Package" association="ex1">
      <type xmi:type="uml:Class" href="pathmap://UML_METAMODELS/UML.metamodel.uml#Package"/>
    </ownedAttribute>
    <ownedAttribute xmi:id="a3" name="version">
      <type xmi:type="uml:PrimitiveType" href="pathmap://UML_LIBRARIES/UMLPrimitiveTypes.library.uml#String"/>
    </ownedAttribute>
  </packagedElement>
  <packagedElement xmi:type="uml:Stereotype" xmi:id="s2" name="Leaf">
    <ownedRule xmi:id="a4" name="NoSubpackages" constrainedElement="s2">
      <specification xmi:type="uml:OpaqueExpression" xmi:id="sp2">
        <language>OCL2.0</language>
        <body>inv: self.base_Package.nestedPackage->size()=0</body>
      </specification>
    </ownedRule>
    <ownedAttribute xmi:id="a5" name="base_Package" association="ex2">
      <type xmi:type="uml:Class" href="pathmap://UML_METAMODELS/UML.metamodel.uml#Package"/>
    </ownedAttribute>
  </packagedElement>
  <packagedElement xmi:type="uml:Stereotype" xmi:id="s3" name="FeatureType">
    <ownedAttribute xmi:id="a6" name="base_Class" association="ex3">
      <type xmi:type="uml:Class" href="pathmap://UML_METAMODELS/UML.metamodel.uml#Class"/>
    </ownedAttribute>
    <ownedAttribute xmi:id="a7" name="isCollection">
      <type xmi:type="uml:PrimitiveType" href="pathmap://UML_LIBRARIES/UMLPrimitiveTypes.library.uml#Boolean"/>
    </ownedAttribute>
  </packagedElement>
  <packagedElement xmi:type="uml:Stereotype" xmi:id="s4" name="Union">
    <ownedRule xmi:id="a8" name="NumberOfAttributes" constrainedElement="s4">
      <specification xmi:type="uml:OpaqueExpression" xmi:id="sp3">
```

```
        <language>OCL2.0</language>
        <body>
          inv: self.base_DataType.ownedAttribute->size()>=2&#xD;inv: self.base_DataType.ownedAttribute->forAll(c |
              c.lower=0 and c.upper=1)&#xD;inv: self.base_DataType.ownedRule->notEmpty()
        </body>
      </specification>
    </ownedRule>
    <ownedAttribute xmi:id="a9" name="base_DataType" association="ex4">
      <type xmi:type="uml:Class" href="pathmap://UML_METAMODELS/UML.metamodel.uml#DataType"/>
    </ownedAttribute>
  </packagedElement>
  <packagedElement xmi:type="uml:Stereotype" xmi:id="s5" name="CodeList">
    <ownedAttribute xmi:id="a10" name="base_Enumeration" association="ex5">
      <type xmi:type="uml:Class" href="pathmap://UML_METAMODELS/UML.metamodel.uml#Enumeration"/>
    </ownedAttribute>
  </packagedElement>
  <packagedElement xmi:type="uml:Stereotype" xmi:id="s6" name="Property">
    <ownedAttribute xmi:id="a11" name="base_Property" association="ex6">
      <type xmi:type="uml:Class" href="pathmap://UML_METAMODELS/UML.metamodel.uml#Property"/>
    </ownedAttribute>
  </packagedElement>
  <packagedElement xmi:type="uml:Extension" xmi:id="ex1" name="A_Package_ApplicationSchema" memberEnd="ee1
      a2">
    <ownedEnd xmi:type="uml:ExtensionEnd" xmi:id="ee1" name="extension_ApplicationSchema" type="s1"
        aggregation="composite" association="ex1"/>
  </packagedElement>
  <packagedElement xmi:type="uml:Extension" xmi:id="ex2" name="A_Package_Leaf" memberEnd="ee2 a5">
    <ownedEnd xmi:type="uml:ExtensionEnd" xmi:id="ee2" name="extension_Leaf" type="s2" aggregation="composite"
        association="ex2"/>
  </packagedElement>
  <packagedElement xmi:type="uml:Extension" xmi:id="ex3" name="A_Class_FeatureType" memberEnd="ee3 a6">
    <ownedEnd xmi:type="uml:ExtensionEnd" xmi:id="ee3" name="extension_FeatureType" type="s3"
        aggregation="composite" association="ex3"/>
  </packagedElement>
  <packagedElement xmi:type="uml:Extension" xmi:id="ex4" name="A_DataType_Union" memberEnd="ee4 a9">
    <ownedEnd xmi:type="uml:ExtensionEnd" xmi:id="ee4" name="extension_Union" type="s4" aggregation="composite"
        association="ex4"/>
  </packagedElement>
  <packagedElement xmi:type="uml:Extension" xmi:id="ex5" name="A_Enumeration_CodeList" memberEnd="ee5 a10">
    <ownedEnd xmi:type="uml:ExtensionEnd" xmi:id="ee5" name="extension_CodeList" type="s5"
        aggregation="composite" association="ex5"/>
  </packagedElement>
  <packagedElement xmi:type="uml:Extension" xmi:id="ex6" name="A_Property_Property" memberEnd="ee6 a11">
    <ownedEnd xmi:type="uml:ExtensionEnd" xmi:id="ee6" name="extension_Property" type="s6"
        aggregation="composite" association="ex6"/>
  </packagedElement>
</uml:Profile>
```

D Implementation of the multi-level information integration framework

This appendix contains further documentation on the implementation of the multi-level information integration framework which is described in chapter 7, page 139. The additional documentation includes the ATL code which maps the INSPIRE UML profile to the Core UML profile, the contents of the UMLT diagram which defines the transformation between the AAA UML model and the INSPIRE UML model, the UMLT diagram which defines the transformation between the DFK UML model and the INSPIRE UML model as well as differences regarding the XMI structure generated by the UML tools Enterprise Architect and Eclipse UML2.

D.1 Transformation between the AAA application schema and the INSPIRE themes Cadastral Parcels and Administrative Units

D.1.1 ATL transformation definition between the INSPIRE UML profile and the Core UML profile

Listing D.1 contains the ATL code of the transformation module *INSPIRE2Core* for mapping the INSPIRE UML profile to the Core UML profile which was implemented based on the mappings defined in table 7.3, page 145. The code makes use of superimposition (cf. section 3.4.1, page 43) by superimposing the transformation module INSPIRE2Core on top of the transformation module *Copy2UML* from (Wagelaar 2010). The Copy2UML module provides transformation rules for every metaclass in the UML 2 metamodel by simply copying the source model elements into corresponding target model elements. These transformation rules are extended by the additional rules defined in the INSPIRE2Core module.

Listing D.1: The implemented ATL code for mapping the INSPIRE UML profile to the Core UML profile

```
-- @nsURI UML2=http://www.eclipse.org/uml2/3.0.0/UML

module INSPIRE2Core;
create OUT: UML2 from IN: UML2, COREPROFILE: UML2, STANDARDPROFILE: UML2;

uses "lib::UML2";
uses UML2Copy;

helper def: coreUmlProfile: UML2!"uml::Profile" =
  'CoreUMLProfile'.profile().debug('coreUmlProfile');

helper def: standardUmlProfile: UML2!"uml::Profile" =
  'Standard'.profile().debug('standardUmlProfile');

helper context UML2!Element def: hasStereotype(name: String): Boolean =
```

```
  not self.getAppliedStereotype(name).oclIsUndefined();

helper def: getVoidReasonValueClass: UML2!"uml::Class" =
  thisModule.getClass(UML2!"uml::Class".allInstancesFrom('IN'), 'VoidReasonValue');

helper def: getClass(s: Sequence(UML2!"uml::Class"), className: String): UML2!"uml::Class" =
  s -> any(e | e.name = className);

helper def: existsClass(s: Sequence(UML2!"uml::Class"), className: String): UML2!"uml::Class" =
  s -> exists(e | e.name = className);

rule Model {
  from
    s: UML2!"uml::Model" in IN
  using {
    voidablePackage: UML2!"uml::Package" = '';
  }
  to
    t: UML2!"uml::Model" (
      __xmiID__ <- s.__xmiID__,
      name <- s.name.debug('Model'),
      visibility <- s.visibility,
      ownedComment <- s.ownedComment,
      clientDependency <- s.clientDependency,
      elementImport <- s.elementImport,
      packageImport <- s.packageImport,
      ownedRule <- s.ownedRule,
      packageMerge <- s.packageMerge,
      packagedElement <- s.packagedElement
    )
  do {
    t.applyProfile(thisModule.coreUmlProfile);
    t.applyProfile(thisModule.standardUmlProfile);
    voidablePackage <- thisModule.createVoidablePackage(t);
    t.packagedElement <- s.packagedElement -> append(voidablePackage);
  }
}

rule createVoidablePackage(model: UML2!"uml::Model") {
  to
    t: UML2!"uml::Package" (
      name <- 'VoidableClasses'
    )
  do {
    t;
  }
}

rule Package {
  from
    s: UML2!"uml::Package" in IN (
      s.oclIsTypeOf(UML2!"uml::Package")
    )
  to
    t: UML2!"uml::Package" (
      __xmiID__ <- s.__xmiID__,
      name <- s.name,
      visibility <- s.visibility,
      ownedComment <- s.ownedComment,
      clientDependency <- s.clientDependency,
      elementImport <- s.elementImport,
      packageImport <- s.packageImport,
      ownedRule <- s.ownedRule
      packageMerge <- s.packageMerge,
      packagedElement <- s.packagedElement
```

```
    )
}

rule Class {
  from
    s: UML2!"uml::Class" in IN (
      s.oclIsTypeOf(UML2!"uml::Class") and not s.hasStereotype('UMLProfileforINSPIREdataspecifications::codeList')
            and not s.hasStereotype('UMLProfileforINSPIREdataspecifications::dataType')
    )
  to
    t: UML2!"uml::Class" (
      __xmiID__ <- s.__xmiID__,
      name <- s.name,
      visibility <- s.visibility,
      isLeaf <- s.isLeaf,
      isAbstract <- s.isAbstract,
      ownedComment <- s.ownedComment,
      clientDependency <- s.clientDependency,
      elementImport <- s.elementImport,
      packageImport <- s.packageImport,
      ownedRule <- s.ownedRule,
      generalization <- s.generalization,
      substitution <- s.substitution,
      ownedAttribute <- s.ownedAttribute,
      ownedBehavior <- s.ownedBehavior,
      interfaceRealization <- s.interfaceRealization,
      nestedClassifier <- s.nestedClassifier,
      ownedOperation <- s.ownedOperation
    )
}

-- Mapping from stereotype «applicationSchema»/UML metaclass Package to stereotype «ApplicationSchema»/UML
      metaclass Package
rule ApplicationSchema extends Package {
  from
    s: UML2!"uml::Package" in IN (
      s.oclIsTypeOf(UML2!"uml::Package") and
            s.hasStereotype('UMLProfileforINSPIREdataspecifications::applicationSchema')
    )
  to
    t: UML2!"uml::Package" (
  )
  do {
    t.applyStereotype('ApplicationSchema'.stereotype());
  }
}

-- Mapping from stereotype «leaf»/UML metaclass Package to stereotype «Leaf»/UML metaclass Package
rule Leaf extends Package {
  from
    s: UML2!"uml::Package" in IN (
      s.oclIsTypeOf(UML2!"uml::Package") and s.hasStereotype('UMLProfileforINSPIREdataspecifications::leaf')
    )
  to
    t: UML2!"uml::Package" (
  )
  do {
    t.applyStereotype('Leaf'.stereotype());
  }
}

-- Mapping from stereotype «featureType»/UML metaclass Class to stereotype «FeatureType»/UML metaclass Class
rule FeatureType extends Class {
  from
    s: UML2!"uml::Class" in IN (
```

```
        s.oclIsTypeOf(UML2!"uml::Class") and s.hasStereotype('UMLProfileforINSPIREdataspecifications::featureType')
    )
  to
    t: UML2!"uml::Class" (
      ownedAttribute <- s.ownedAttribute -> collect(e | thisModule.VoidableAttributes(e))
    )
  do {
    t.applyStereotype('FeatureType'.stereotype());
  }
}

-- Mapping from stereotype «voidable»/UML metaclass Property to specific voidable classes
unique lazy rule VoidableAttributes {
  from
    s: UML2!"uml::Property" in IN
  using {
    classInstances: Sequence(UML2!"uml::Class") = UML2!"uml::Class".allInstancesFrom('OUT');
  }
  to
    t: UML2!"uml::Property" (
      __xmiID__ <- s.__xmiID__,
      name <- s.name,
      type <- if s.hasStereotype('UMLProfileforINSPIREdataspecifications::voidable')
                then
                  if thisModule.existsClass(classInstances, 'Voidable' + s.type.name)
                    then thisModule.getClass(classInstances, 'Voidable' + s.type.name)
                    else thisModule.CreateVoidableClass(s.type)
                  endif
                else s.type
              endif,
      lowerValue <- s.lowerValue,
      upperValue <- s.upperValue,
      association <- s.association
    )
}
lazy rule CreateVoidableClass {
  from
    s: UML2!"uml::Type" in IN
  to
    voidableClass: UML2!"uml::Class" (
      name <- 'Voidable' + s.name,
      ownedAttribute <- Sequence{firstAttribute,secondAttribute}
    ),
    firstAttribute: UML2!"uml::Property" (
      name <- s.name.toLower(),
      type <- s
    ),
    secondAttribute: UML2!"uml::Property" (
      name <- 'voidReason',
      type <- thisModule.getVoidReasonValueClass
    )
}

-- Mapping from stereotype «placeholder»/UML metaclass Class to stereotype «FeatureType»/UML metaclass Class
rule Placeholder extends Class {
  from
    s: UML2!"uml::Class" in IN (
      s.oclIsTypeOf(UML2!"uml::Class") and s.hasStereotype('UMLProfileforINSPIREdataspecifications::placeholder')
    )
  to
    t: UML2!"uml::Class" (
  )
  do {
    t.applyStereotype('FeatureType'.stereotype());
  }
```

```
}

-- Mapping from stereotype «type»/UML metaclass Class to stereotype «type»/UML metaclass Class
rule Type extends Class {
  from
    s: UML2!"uml::Class" in IN (
      s.oclIsTypeOf(UML2!"uml::Class") and s.hasStereotype('UMLProfileforINSPIREdataspecifications::type')
    )
  to
    t: UML2!"uml::Class" (
  )
  do {
    t.applyStereotype('Type'.stereotype());
  }
}

-- Mapping from stereotype «union»/UML metaclass Class to stereotype «Union»/UML metaclass DataType
rule Union extends Class {
  from
    s: UML2!"uml::Class" in IN (
      s.oclIsTypeOf(UML2!"uml::Class") and s.hasStereotype('UMLProfileforINSPIREdataspecifications::union')
    )
  using {
    constraintValue: String = '';
  }
  to
    t: UML2!"uml::DataType" (
      ownedAttribute <- s.ownedAttribute -> collect(e | thisModule.SetMultiplicity(e)),
      ownedRule <- getRule
    ),
    getRule: UML2!"uml::Constraint" (
      name <- s.name + '_Constraint',
      constrainedElement <- t,
      specification <- getSpec
    ),
    getSpec: UML2!"uml::OpaqueExpression" (
      language <- 'OCL2.0'
    )
  do {
    for(p in s.ownedAttribute){
      constraintValue <- constraintValue.concat('self.' + p.name + ' xor ');
    }
    constraintValue <- constraintValue.substring(1, constraintValue.size() - 5);
    t.ownedRule -> first().specification.body <- constraintValue;
    t.applyStereotype('Union'.stereotype());
  }
}
lazy rule SetMultiplicity {
  from
    s: UML2!"uml::Property" in IN
  to
    t: UML2!"uml::Property" (
      __xmiID__ <- s.__xmiID__,
      name <- s.name,
      type <- s.type,
      lowerValue <- getLower,
      upperValue <- getUpper
    ),
    getLower: UML2!"uml::LiteralInteger" (
      value <- 0
    ),
    getUpper: UML2!"uml::LiteralInteger" (
      value <- 1
    )
}
```

```
-- Mapping from stereotype «codeList»/UML metaclass Class to stereotype «CodeList»/UML metaclass Enumeration
rule CodeList {
  from
    s: UML2!"uml::Class" in IN (
      s.oclIsTypeOf(UML2!"uml::Class") and s.hasStereotype('UMLProfileforINSPIREdataspecifications::codeList')
    )
  to
    t: UML2!"uml::Enumeration" (
      __xmiID__ <- s.__xmiID__,
      name <- s.name,
      visibility <- s.visibility,
      ownedLiteral <- s.ownedAttribute -> collect(e | thisModule.Property2EnumerationLiteral(e))
    )
  do {
    t.applyStereotype('CodeList'.stereotype());
  }
}
lazy rule Property2EnumerationLiteral {
  from
    s: UML2!"uml::Property"
  to
    t: UML2!"uml::EnumerationLiteral" (
      __xmiID__ <- s.__xmiID__,
      name <- s.name + '=' + s.defaultValue.value,
      visibility <- s.visibility
    )
}

-- Mapping from stereotype «dataType»/UML metaclass Class to UML metaclass DataType
rule DataType {
  from
    s: UML2!"uml::Class" in IN (
      s.oclIsTypeOf(UML2!"uml::Class") and s.hasStereotype('UMLProfileforINSPIREdataspecifications::dataType')
    )
  to
    t: UML2!"uml::DataType" (
      __xmiID__ <- s.__xmiID__,
      name <- s.name,
      visibility <- s.visibility,
      isLeaf <- s.isLeaf,
      isAbstract <- s.isAbstract,
      ownedComment <- s.ownedComment,
      ownedRule <- s.ownedRule,
      generalization <- s.generalization,
      ownedAttribute <- s.ownedAttribute -> collect(e | thisModule.VoidableAttributes(e)),
      ownedOperation <- s.ownedOperation
    )
}
```

D.1.2 UMLT transformation definition between AAA and INSPIRE

The UMLT diagram in figure 7.2, page 154, shows a general overview of the UMLT transformation definition created between the AAA application schema and the INSPIRE themes Cadastral Parcels and Administrative Units. The diagram contains several TransformationValueMaps and VirtualAssociations as well as the TransformationActions FSGeometryConverter and AAA2INSPIRE. For purposes of clarity regarding the page layout, they are only represented by correspondingly named rectangles in the diagram. The detailed contents of these rectangles are listed in the following.

Gebietsgrenze_ValueMap
7101=1stOrder
7102=2ndOrder
7103=3rdOrder
7104=4thOrder
7105=5thOrder
7106=6thOrder

BesondereFlurstuecksgrenze_ValueMap
7101=1stOrder
7102=2ndOrder
7103=3rdOrder
7104=4thOrder
7105=5thOrder
7106=6thOrder

Figure D.1: TransformationValueMaps from the UMLT transformation definition between AAA and INSPIRE

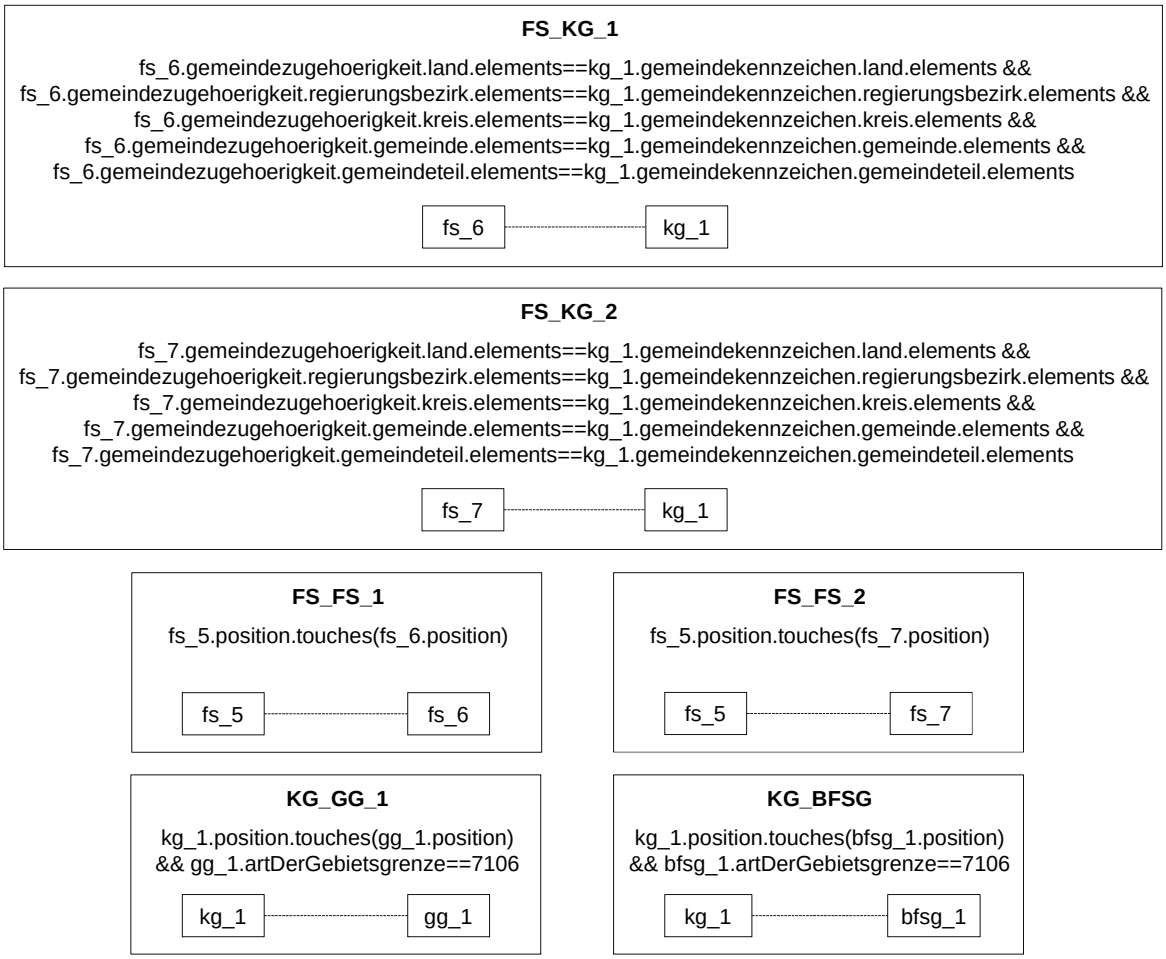


Figure D.2: VirtualAssociations from the UMLT transformation definition between AAA and INSPIRE

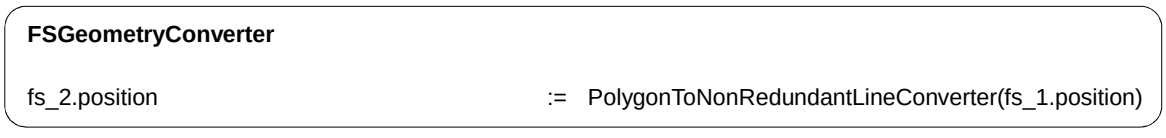


Figure D.3: TransformationAction *FSGeometryConverter* from the UMLT transformation definition between AAA and INSPIRE

AAA2INSPIRE

cp_1.geometry	:=	fs_6.position
cp_1.inspireId.localId	:=	fs_6.identifikator.UUID.elements
cp_1.inspireId.namespace	:=	"http://www.adv-online.de/namespaces/adv/gid6.0"
cp_1.inspireId.versionId.characterstring.elements	:=	StringConcatenator(StringConcatenator(fs_6.lebenszeitintervall.beginnt.year.elements, fs_6.lebenszeitintervall.beginnt.month.elements), fs_6.lebenszeitintervall.beginnt.day.elements)
cp_1.label.elements	:=	StringConcatenator(StringConcatenator(fs_6.flurstuecksnummer.zaehler.elements,"/"), fs_6.flurstuecksnummer.nenner.elements)
cp_1.nationalCadastralReference.elements	:=	fs_6.flurstueckskennzeichen.elements
cp_1.areaValue.area.value	:=	fs_6.amtlicheFlaeche.value
cp_1.areaValue.area.uom.uomSymbol.elements	:=	fs_6.amtlicheFlaeche.uom.uomSymbol.elements
cp_1.referencePoint	:=	fs_6.objektkoordinaten
cp_1.beginLifespanVersion.datetime.century.elements	:=	fs_6.lebenszeitintervall.beginnt.century.elements
cp_1.beginLifespanVersion.datetime.year.elements	:=	fs_6.lebenszeitintervall.beginnt.year.elements
cp_1.beginLifespanVersion.datetime.month.elements	:=	fs_6.lebenszeitintervall.beginnt.month.elements
cp_1.beginLifespanVersion.datetime.day.elements	:=	fs_6.lebenszeitintervall.beginnt.day.elements
cp_1.beginLifespanVersion.datetime.precision	:=	fs_6.lebenszeitintervall.beginnt.precision
cp_1.beginLifespanVersion.datetime.hour.elements	:=	fs_6.lebenszeitintervall.beginnt.hour.elements
cp_1.beginLifespanVersion.datetime.minute.elements	:=	fs_6.lebenszeitintervall.beginnt.minute.elements
cp_1.beginLifespanVersion.datetime.second.elements	:=	fs_6.lebenszeitintervall.beginnt.second.elements
cp_1.beginLifespanVersion.datetime.timeZone.elements	:=	fs_6.lebenszeitintervall.beginnt.timeZone.elements
cb_1.geometry	:=	fs_5.position
cb_1.inspireId.localId	:=	UUIDGenerator()
cb_1.inspireId.namespace	:=	"http://www.adv-online.de/namespaces/adv/gid6.0"
cb_1.inspireId.versionId.characterstring.elements	:=	StringConcatenator(StringConcatenator(fs_6.lebenszeitintervall.beginnt.year.elements, fs_6.lebenszeitintervall.beginnt.month.elements), fs_6.lebenszeitintervall.beginnt.day.elements)
cb_1.beginLifespanVersion.datetime.century.elements	:=	fs_6.lebenszeitintervall.beginnt.century.elements
cb_1.beginLifespanVersion.datetime.year.elements	:=	fs_6.lebenszeitintervall.beginnt.year.elements
cb_1.beginLifespanVersion.datetime.month.elements	:=	fs_6.lebenszeitintervall.beginnt.month.elements
cb_1.beginLifespanVersion.datetime.day.elements	:=	fs_6.lebenszeitintervall.beginnt.day.elements
cb_1.beginLifespanVersion.datetime.precision	:=	fs_6.lebenszeitintervall.beginnt.precision
cb_1.beginLifespanVersion.datetime.hour.elements	:=	fs_6.lebenszeitintervall.beginnt.hour.elements
cb_1.beginLifespanVersion.datetime.minute.elements	:=	fs_6.lebenszeitintervall.beginnt.minute.elements
cb_1.beginLifespanVersion.datetime.second.elements	:=	fs_6.lebenszeitintervall.beginnt.second.elements
cb_1.beginLifespanVersion.datetime.timeZone.elements	:=	fs_6.lebenszeitintervall.beginnt.timeZone.elements
cb_1.estimatedAccuracy.length.value	:=	???
cb_1.estimatedAccuracy.length.uom.uomSymbol.elements	:=	"m"
cp_2.geometry	:=	fs_7.position
cp_2.inspireId.localId	:=	fs_7.identifikator.UUID.elements
cp_2.inspireId.namespace	:=	"http://www.adv-online.de/namespaces/adv/gid6.0"
cp_2.inspireId.versionId.characterstring.elements	:=	StringConcatenator(StringConcatenator(fs_7.lebenszeitintervall.beginnt.year.elements, fs_7.lebenszeitintervall.beginnt.month.elements), fs_7.lebenszeitintervall.beginnt.day.elements)
cp_2.label.elements	:=	fs_7.flurstuecksnummer.zaehler.elements
cp_2.nationalCadastralReference.elements	:=	fs_7.flurstueckskennzeichen.elements
cp_2.areaValue.area.value	:=	fs_7.amtlicheFlaeche.value
cp_2.areaValue.area.uom.uomSymbol.elements	:=	fs_7.amtlicheFlaeche.uom.uomSymbol.elements
cp_2.referencePoint	:=	fs_7.objektkoordinaten
…		

Figure D.4: TransformationAction *AAA2INSPIRE* from the UMLT transformation definition between AAA and INSPIRE – Part 1

```
...
cp_2.beginLifespanVersion.datetime.century.elements      :=  fs_7.lebenszeitintervall.beginnt.century.elements
cp_2.beginLifespanVersion.datetime.year.elements         :=  fs_7.lebenszeitintervall.beginnt.year.elements
cp_2.beginLifespanVersion.datetime.month.elements        :=  fs_7.lebenszeitintervall.beginnt.month.elements
cp_2.beginLifespanVersion.datetime.day.elements          :=  fs_7.lebenszeitintervall.beginnt.day.elements
cp_2.beginLifespanVersion.datetime.precision             :=  fs_7.lebenszeitintervall.beginnt.precision
cp_2.beginLifespanVersion.datetime.hour.elements         :=  fs_7.lebenszeitintervall.beginnt.hour.elements
cp_2.beginLifespanVersion.datetime.minute.elements       :=  fs_7.lebenszeitintervall.beginnt.minute.elements
cp_2.beginLifespanVersion.datetime.second.elements       :=  fs_7.lebenszeitintervall.beginnt.second.elements
cp_2.beginLifespanVersion.datetime.timeZone.elements     :=  fs_7.lebenszeitintervall.beginnt.timeZone.elements

cb_2.geometry                                            :=  fs_5.position
cb_2.inspireId.localId                                   :=  UUIDGenerator()
cb_2.inspireId.namespace                                 :=  "http://www.adv-online.de/namespaces/adv/gid6.0"
cb_2.inspireId.versionId.characterstring.elements        :=  StringConcatenator(StringConcatenator(
                                                             fs_7.lebenszeitintervall.beginnt.year.elements,
                                                             fs_7.lebenszeitintervall.beginnt.month.elements),
                                                             fs_7.lebenszeitintervall.beginnt.day.elements)
cb_2.beginLifespanVersion.datetime.century.elements      :=  fs_7.lebenszeitintervall.beginnt.century.elements
cb_2.beginLifespanVersion.datetime.year.elements         :=  fs_7.lebenszeitintervall.beginnt.year.elements
cb_2.beginLifespanVersion.datetime.month.elements        :=  fs_7.lebenszeitintervall.beginnt.month.elements
cb_2.beginLifespanVersion.datetime.day.elements          :=  fs_7.lebenszeitintervall.beginnt.day.elements
cb_2.beginLifespanVersion.datetime.precision             :=  fs_7.lebenszeitintervall.beginnt.precision
cb_2.beginLifespanVersion.datetime.hour.elements         :=  fs_7.lebenszeitintervall.beginnt.hour.elements
cb_2.beginLifespanVersion.datetime.minute.elements       :=  fs_7.lebenszeitintervall.beginnt.minute.elements
cb_2.beginLifespanVersion.datetime.second.elements       :=  fs_7.lebenszeitintervall.beginnt.second.elements
cb_2.beginLifespanVersion.datetime.timeZone.elements     :=  fs_7.lebenszeitintervall.beginnt.timeZone.elements
cb_2.estimatedAccuracy.length.value                      :=  ???
cb_2.estimatedAccuracy.length.uom.uomSymbol.elements     :=  "m"

au_1.geometry                                            :=  kg_1.position
au_1.nationalCode                                        :=  kg_1.schluesselGesamt.elements
au_1.inspireId.localId                                   :=  kg_1.identifikator.UUID.elements
au_1.inspireId.namespace                                 :=  "http://www.adv-online.de/namespaces/adv/gid6.0"
au_1.inspireId.versionId.characterstring.elements        :=  StringConcatenator(StringConcatenator(
                                                             kg_1.lebenszeitintervall.beginnt.year.elements,
                                                             kg_1.lebenszeitintervall.beginnt.month.elements),
                                                             kg_1.lebenszeitintervall.beginnt.day.elements)
au_1.nationalLevel                                       :=  "6thOrder"
au_1.country                                             :=  "DE“
au_1.residenceOfAuthority.voidReason                     :=  "Unpopulated"
au_1.beginLifespanVersion.datetime.century.elements      :=  kg_1.lebenszeitintervall.beginnt.century.elements
au_1.beginLifespanVersion.datetime.year.elements         :=  kg_1.lebenszeitintervall.beginnt.year.elements
au_1.beginLifespanVersion.datetime.month.elements        :=  kg_1.lebenszeitintervall.beginnt.month.elements
au_1.beginLifespanVersion.datetime.day.elements          :=  kg_1.lebenszeitintervall.beginnt.day.elements
au_1.beginLifespanVersion.datetime.precision             :=  kg_1.lebenszeitintervall.beginnt.precision
au_1.beginLifespanVersion.datetime.hour.elements         :=  kg_1.lebenszeitintervall.beginnt.hour.elements
au_1.beginLifespanVersion.datetime.minute.elements       :=  kg_1.lebenszeitintervall.beginnt.minute.elements
au_1.beginLifespanVersion.datetime.second.elements       :=  kg_1.lebenszeitintervall.beginnt.second.elements
au_1.beginLifespanVersion.datetime.timeZone.elements     :=  kg_1.lebenszeitintervall.beginnt.timeZone.elements
au_1.NUTS.voidReason                                     :=  "Unpopulated"

ab_1.geometry                                            :=  gg_1.position
ab_1.inspireId.localId                                   :=  gg_1.identifikator.UUID.elements
ab_1.inspireId.namespace                                 :=  http://www.adv-online.de/namespaces/adv/gid6.0
...
```

Figure D.5: TransformationAction *AAA2INSPIRE* from the UMLT transformation definition between AAA and INSPIRE – Part 2

```
...
ab_1.inspireId.versionId.characterstring.elements := StringConcatenator(StringConcatenator(
                                                     gg_1.lebenszeitintervall.beginnt.year.elements,
                                                     gg_1.lebenszeitintervall.beginnt.month.elements),
                                                     gg_1.lebenszeitintervall.beginnt.day.elements)
ab_1.country := "DE"
ab_1.nationalLevel := ValueMapper(gg_1.artDerGebietsgrenze,
                      "Gebietsgrenze_ValueMap")
ab_1.legalStatus.legalstatusvalue := "agreed"
ab_1.technicalStatus.technicalstatusvalue := "edgeMatched"
ab_1.beginLifespanVersion.datetime.century.elements := gg_1.lebenszeitintervall.beginnt.century.elements
ab_1.beginLifespanVersion.datetime.year.elements := gg_1.lebenszeitintervall.beginnt.year.elements
ab_1.beginLifespanVersion.datetime.month.elements := gg_1.lebenszeitintervall.beginnt.month.elements
ab_1.beginLifespanVersion.datetime.day.elements := gg_1.lebenszeitintervall.beginnt.day.elements
ab_1.beginLifespanVersion.datetime.precision := gg_1.lebenszeitintervall.beginnt.precision
ab_1.beginLifespanVersion.datetime.hour.elements := gg_1.lebenszeitintervall.beginnt.hour.elements
ab_1.beginLifespanVersion.datetime.minute.elements := gg_1.lebenszeitintervall.beginnt.minute.elements
ab_1.beginLifespanVersion.datetime.second.elements := gg_1.lebenszeitintervall.beginnt.second.elements
ab_1.beginLifespanVersion.datetime.timeZone.elements := gg_1.lebenszeitintervall.beginnt.timeZone.elements

ab_2.geometry := bfsg_1.position
ab_2.inspireId.localId := bfsg_1.identifikator.UUID.elements
ab_2.inspireId.namespace := "http://www.adv-online.de/namespaces/adv/gid6.0"
ab_2.inspireId.versionId.characterstring.elements := StringConcatenator(StringConcatenator(
                                                     bfsg_1.lebenszeitintervall.beginnt.year.elements,
                                                     bfsg_1.lebenszeitintervall.beginnt.month.elements),
                                                     bfsg_1.lebenszeitintervall.beginnt.day.elements)
ab_2.country := "DE"
ab_2.nationalLevel := ValueMapper(bfsg_1.artDerFlurstuecksgrenze,
                      "BesondereFlurstuecksgrenze_ValueMap")
ab_2.legalStatus.legalstatusvalue := "agreed"
ab_2.technicalStatus.technicalstatusvalue := "edgeMatched"
ab_2.beginLifespanVersion.datetime.century.elements := bfsg_1.lebenszeitintervall.beginnt.century.elements
ab_2.beginLifespanVersion.datetime.year.elements := bfsg_1.lebenszeitintervall.beginnt.year.elements
ab_2.beginLifespanVersion.datetime.month.elements := bfsg_1.lebenszeitintervall.beginnt.month.elements
ab_2.beginLifespanVersion.datetime.day.elements := bfsg_1.lebenszeitintervall.beginnt.day.elements
ab_2.beginLifespanVersion.datetime.precision := bfsg_1.lebenszeitintervall.beginnt.precision
ab_2.beginLifespanVersion.datetime.hour.elements := bfsg_1.lebenszeitintervall.beginnt.hour.elements
ab_2.beginLifespanVersion.datetime.minute.elements := bfsg_1.lebenszeitintervall.beginnt.minute.elements
ab_2.beginLifespanVersion.datetime.second.elements := bfsg_1.lebenszeitintervall.beginnt.second.elements
ab_2.beginLifespanVersion.datetime.timeZone.elements := bfsg_1.lebenszeitintervall.beginnt.timeZone.elements
```

Figure D.6: TransformationAction *AAA2INSPIRE* from the UMLT transformation definition between AAA and INSPIRE – Part 3

D.2 Transformation between the DFK application schema and the INSPIRE themes Cadastral Parcels and Administrative Units

Figure D.7 shows the general overview of the UMLT transformation definition created between the DFK application schema and the INSPIRE themes Cadastral Parcels and Administrative Units.

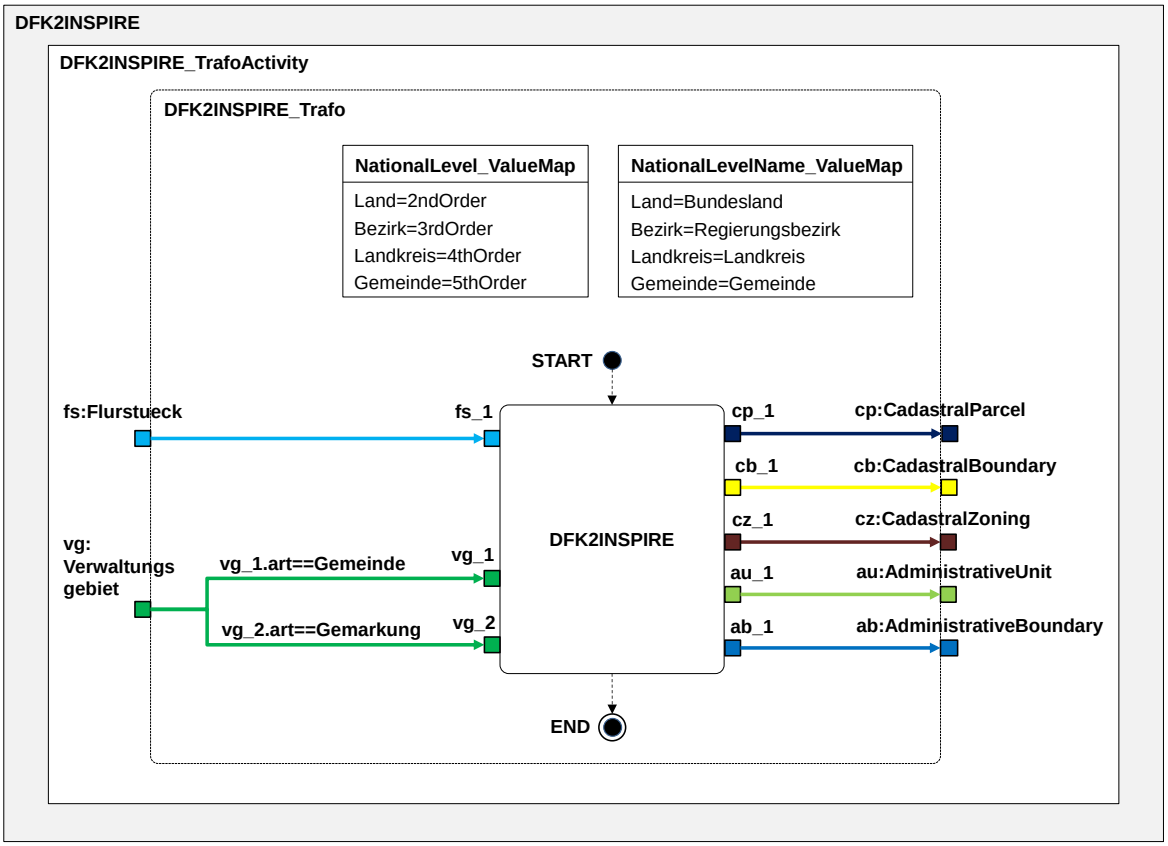


Figure D.7: UMLT transformation definition between the DFK application schema and the INSPIRE themes Cadastral Parcels and Administrative Units

D.3 Differences between Enterprise Architect and Eclipse regarding the XMI structure of UML models extended by UML profiles

The following two listings illustrate the differences regarding the XMI representation of UML models to which UML profiles are applied. Listing D.2 shows the XMI structure created by the UML tool Enterprise Architect (EA) version 10, listing D.3 the XMI structure created by Eclipse UML2 version 3.2.1. The listings are based on XMI version 2.1. The listings only contain those XML elements and XML attributes relevant for illustrating the differences, all other items were omitted for clarity of presentation.

One difference is that EA includes the UML profiles within the `<xmi:Extension>` part of the XMI model document, whereas Eclipse UML2 stores the UML profiles as separate XMI profile documents and applies them to the Eclipse XMI model document by means of the `<profileApplication>` section within the `<uml:Model>` part. In EA, the application of UML profiles takes place in an EA-specific way. Furthermore, the EA XMI model document applies the stereotypes within the `<uml:Model>` part, whereas the Eclipse XMI model document applies the stereotypes following the `<uml:Model>` part. Above that, the Eclipse XMI profile document contains further Ecore-specific information which is not shown here.

Listing D.2: XMI 2.1 structure generated by Enterprise Architect version 10 representing the UML model *InspireModel* to which the UML profile *InspireProfile* is applied and the UML class *CadastralParcel* to which the stereotype «FeatureType» is assigned

```
<?xml version="1.0" encoding="UTF-8"?>
<xmi:XMI xmi:version="2.1" xmlns:InspireProfile="http://www.sparxsystems.com/profiles/InspireProfile/1.0" ...>
  <!-- The actual UML model -->
  <uml:Model xmi:id="m1" xmi:type="uml:Model" name="InspireModel">
    <!-- UML class CadastralParcel -->
    <packagedElement xmi:type="uml:Class" xmi:id="c1" name="CadastralParcel">
      ...
    </packagedElement>
    ...
    <!-- Here the stereotype «FeatureType» is applied to the UML class CadastralParcel -->
    <InspireProfile:FeatureType xmi:id="s1" base_Class="c1" xsdEncodingRule="ISO19136_2007" .../>
  </uml:Model>
  <!-- The extension part containing additional information which is not part of the UML model itself -->
  <xmi:Extension extender="Enterprise Architect" extenderID="6.5">
    ...
    <!-- Here the UML profiles are listed -->
    <profiles>
      <uml:Profile xmi:version="2.1" xmi:id="p1" name="InspireProfile" metamodelReference="mm1" ...>
        ...
      </uml:Profile>
      ...
    </profiles>
    ...
  </xmi:Extension>
</xmi:XMI>
```

Listing D.3: XMI 2.1 structure generated by Eclipse UML2 version 3.2.1 representing the UML model *InspireModel* to which the UML profile *InspireProfile* is applied and the UML class *CadastralParcel* to which the stereotype «FeatureType» is assigned

```
<!-- Eclipse XMI model document -->
<?xml version="1.0" encoding="UTF-8"?>
<xmi:XMI xmi:version="2.1" xmlns:InspireProfile="http://InspireProfile/v1/0" ...>
  <uml:Model xmi:id="m1" xmi:type="uml:Model" name="InspireModel">
    <!-- UML class CadastralParcel -->
    <packagedElement xmi:type="uml:Class" xmi:id="c1" name="CadastralParcel">
      ...
    </packagedElement>
    ...
    <!-- Here the UML profile is applied to the UML model -->
    <profileApplication>
      <eAnnotations>
        <references xmi:type="ecore:EPackage" href="InspireProfile.profile.uml#p1"/>
      </eAnnotations>
      <appliedProfile href="InspireProfile.profile.uml#p1"/>
    </profileApplication>
  </uml:Model>
  <!-- Here the stereotype «FeatureType» is applied to the UML class CadastralParcel -->
  <InspireProfile:FeatureType xmi:id="s1" base_Class="c1" xsdEncodingRule="ISO19136_2007" .../>
  <!-- The extension part -->
  <xmi:Extension>...</xmi:Extension>
</xmi:XMI>

<!-- Eclipse XMI profile document -->
<?xml version="1.0" encoding="UTF-8"?>
<uml:Profile xmi:version="2.1" xmi:id="p1" name="InspireProfile" metamodelReference="mm1" ...>
  ...
</uml:Profile>
```

D.4 Implemented UMLT functions

The following table lists all UMLT functions currently implemented within the FME transformer *UMLTApplier*, which was developed as plug-in for FME as part of the research project mdWFS (cf. section 4.3.3, page 67).

Table D.1: UMLT functions implemented within the FME transformer *UMLTApplier*

UMLT function	Signature
Accumulator	Accumulator(attributeToAnalyze: String, attributeToGroupBy: String, typeOfStatistics: String): float
AreaCalculator	AreaCalculator(geometry: Polygon): int
Dissolver	Dissolver(geometry: Polygon[], groupBy: String): Polygon[]
PointInPolygonExtractor	PointInPolygonExtractor(geometry: Polygon): Point
PolygonToLineConverter	PolygonToLineConverter(geometry: Polygon): Line
PolygonToNonRedundant-LineConverter	PolygonToNonRedundantLineConverter(geometry: Polygon): Line
StringConcatenator	StringConcatenator(sourceAttribute1: String, sourceAttribute2: String): String
SubStringer	SubStringer(sourceAttribute: String, startIndex: int, endIndex: int): String
UUIDGenerator	UUIDGenerator(): String
ValueMapper	ValueMapper(sourceAttribute: String, valueMapName: String): String

Acronyms

AAA	AFIS-ALKIS-ATKIS
ADE	CityGML Application Domain Extension
AdV	Working Committee of the Surveying Authorities of the States of the Federal Republic of Germany (Arbeitsgemeinschaft der Vermessungsverwaltungen der Länder der Bundesrepublik Deutschland)
AFIS	Amtliches Festpunktinformationssystem (German Official Geodetic Control Stations Information System)
AIXM	Aeronautical Information Exchange Model
ALKIS	Amtliches Liegenschaftskatasterinformationssystem (German Official Real Estate Cadastre Information System)
API	Application Programming Interface
ATKIS	Amtliches Topographisch-Kartographisches Informationssystem (German Official Topographic Cartographic Information System)
AU	INSPIRE theme Administrative Units
B-rep	Boundary Representation
BEV	Bundesamt für Eich- und Vermessungswesen (Austrian Federal Office of Metrology and Surveying)
BKG	Bundesamt für Kartographie und Geodäsie (German Federal Agency for Cartography and Geodesy)
CIM	Computation Independent Model
CityGML	City Geography Markup Language
CP	INSPIRE theme Cadastral Parcels
CQL	Common Query Language
CSG	Constructive Solid Geometry
CSL	Conceptual Schema Language
CST	HUMBOLDT Conceptual Schema Translation Service
CSV	Comma-separated values
DKM	Digitale Kadastralmappe (Austrian Digital Cadastral Map)
DLM	Digital Landscape Model

EA	Enterprise Architect
EC	European Commission
EDOAL	Expressive and Declarative Ontology Alignment Language
EGN	EuroGeoNames
EMF	Eclipse Modeling Framework
ESDIN	European Spatial Data Infrastructure with a Best Practice Network
ETH Zurich	Eidgenössische Technische Hochschule Zürich (Swiss Federal Institute of Technology Zurich)
EU	European Union
EuroSDR	European Spatial Data Research
FAS-X	Translating Feature Access Service
FDS	FUSION Data Service
FFS	FME Feature Store
FME	Feature Manipulation Engine
FTP	File Transfer Protocol
GCM	INSPIRE Generic Conceptual Model
GeoInfoDok	Dokumentation zur Modellierung der Geoinformationen des amtlichen Vermessungswesens (Documentation on the Modelling of Geoinformation of Official Surveying and Mapping in Germany)
GFM	General Feature Model
GG25	Gemeindegrenzen 25 (Municipality boundaries of Switzerland)
GI	Geographic Information
GiMoDig	Geospatial Info-Mobility Service by Real-Time Data-Integration and Generalisation
GMF	Eclipse Graphical Modeling Framework
GML	Geography Markup Language
gOML	geographic OML
GUI	Graphical User Interface
HALE	HUMBOLDT Alignment Editor
HML	HUMBOLDT Modelling Language
HUTN	Human-Usable Textual Notation
IDE	Integrated Development Environment
IEC	International Electrotechnical Commission
INSPIRE	Infrastructure for Spatial Information in Europe

INSPIRE KEN	INSPIRE Knowledge Exchange Network
ISO	International Organisation for Standardisation
ISO/DIS	ISO Draft International Standard
ISO/TS	ISO Technical Specification
JRC	Joint Research Centre
KML	Keyhole Markup Language
LDBV	Landesamt für Digitalisierung, Breitband und Vermessung (Bavarian Agency for Digitisation, High-Speed Internet and Surveying)
LGL BW	Landesamt für Geoinformation und Landentwicklung Baden-Württemberg (State Agency for Spatial Information and Rural Development Baden-Wuerttemberg)
LOD	Level of Detail
MADS	Modeling of Application Data with Spatio-temporal Features
MDA	Model-driven Architecture
MDD	Model-driven Development
MDE	Model-driven Engineering
MDSD	Model-driven Software Development
mdWFS	Model-driven Web Feature Service
MOF	Meta Object Facility
NAS	Normbasierte Austauschschnittstelle (Standards-based Data Exchange Interface defined in the GeoInfoDok)
NMCA	National Mapping and Cadastral Agency
OAS	ORCHESTRA Application Schema
OCL	Object Constraint Language
OFS	ORCHESTRA Feature Set
OGC	Open Geospatial Consortium
OMG	Object Management Group
OML	Ontology Mapping Language
OMM	ORCHESTRA Meta-Model
OMT-G	Object Modeling Technique for Geographic Applications
OO	Object-oriented
OOA	Object-oriented Analysis
ORCHESTRA	Open Architecture and Spatial Data Infrastructure for Risk Management

OWL	Web Ontology Language
PIM	Platform Independent Model
PSM	Platform Specific Model
QVT	Query/View/Transformation
RDF	Resource Description Framework
RIF	Rule Interchange Format
SDI	Spatial Data Infrastructure
SMS	Schema Mapping Service
SQL	Structured Query Language
TC211	ISO Technical Committee 211
TGG	Triple Graph Grammar
TLM	Topographisches Landschaftsmodell
TNS	INSPIRE Transformation Network Service
TR	Technical Report
TUM	Technische Universität München
UML	Unified Modeling Language
UMLT	UML Transformations
UoM	Universe of Discourse
URI	Unique Resource Identifier
W3C	World Wide Web Consortium
WFS	OGC Web Feature Service
WPS	OGC Web Processing Service
XMI	XML Metadata Interchange
XML	Extensible Markup Language
XSD	XML Schema Definition
XSL	Extensible Stylesheet Language
XSLT	XSL Transformations

Bibliography

AdV (2009). *Documentation on the Modelling of Geoinformation of Official Surveying and Mapping (GeoInfoDoc), Version 6.0.1*. Working Committee of the Surveying Authorities of the States of the Federal Republic of Germany.

AdV (2014). *Dokumentation zur Modellierung der Geoinformationen des amtlichen Vermessungswesens (GeoInfoDok), Version 7.0.1*. Arbeitsgemeinschaft der Vermessungsverwaltungen der Länder der Bundesrepublik Deutschland (AdV).

AdV (2015). *Dokumente der aktuellen GeoInfoDok-Version 6.0*. URL: http://www.adv-online.de/AAA-Modell/Dokumente-der-GeoInfoDok/GeoInfoDok-6-0/ (visited on 31/08/2015).

Afflerbach, S., Illert, A. and Sarjakoski, T. (2004). 'The Harmonisation Challenge of Core National Topographic Databases in the EU-Project GiMoDig'. In: *XXth ISPRS Congress*. Vol. XXXV. The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences. ISPRS, pp. 129–134.

Atkinson, C., Gutheil, M. and Kiko, K. (2006). 'On the relationship of Ontologies and Models'. In: *Meta-Modelling and Ontologies. Proceedings of the 2nd International Workshop on Meta-Modelling, WoMM 2006, October 12-13, 2006, Karlsruhe, Germany*. Ed. by Brockmans, S., Jung, J. and Sure, Y. Vol. P-96. Lecture Notes in Informatics - Proceedings. Gesellschaft für Informatik, pp. 47–60.

Atkinson, C. and Kühne, T. (2000). 'Strict Profiles: Why and How'. In: *«UML» 2000 — The Unified Modeling Language. Advancing the Standard Third International Conference York, UK, October 2–6, 2000 Proceedings*. Ed. by Evans, A., Kent, S. and Selic, B. Vol. 1939. Lecture Notes in Computer Science. Berlin Heidelberg: Springer, pp. 309–322.

Balley, S., Bucher, B. and Libourel, T. (2006). 'A Service to Customize the Structure of a Geographic Dataset'. In: *On the Move to Meaningful Internet Systems 2006: OTM 2006 Workshops*. Ed. by Meersman, R., Tari, Z. and Herrero, P. Vol. 4278. Lecture Notes in Computer Science. Berlin Heidelberg: Springer, pp. 1703–1711.

Beare, M., Howard, M., Payne, S. and Watson, P. (2010). *Development of Technical Guidance for the INSPIRE Transformation Network Service – State Of The Art Analysis*. RSW Geomatics, 1Spatial and Rob Walker Consultancy.

Beare, M., Payne, S. and Sunderland, R. (2010). *Prototype Report for the INSPIRE Schema Transformation Network Service*. RSW Geomatics, 1Spatial and Rob Walker Consultancy.

BEV (2008a). *DOKUMENTATION, DKM – Shapeformat – Schnittstelle V1.0*. Report. Bundesamt für Eich- und Vermessungswesen.

BEV (2008b). *Satzspiegel für die Grundstücksdaten, Schnittstellenbeschreibung, Version 1.4*. Report. Bundesamt für Eich- und Vermessungswesen.

Bézivin, J. (2005). 'On the unification power of models'. In: *Software & Systems Modeling* 4.2, pp. 171–188.

Bézivin, J. (2006). 'Model Driven Engineering: An Emerging Technical Space'. In: *Generative and Transformational Techniques in Software Engineering*. Ed. by Lämmel, R., Saraiva, J. and Visser, J. Vol. 4143. Lecture Notes in Computer Science. Berlin Heidelberg: Springer, pp. 36–64.

Bishr, Y. (1998). 'Overcoming the semantic and other barriers to GIS interoperability'. In: *International Journal of Geographical Information Science* 12.4, pp. 299–314.

Boole, G. (1854). *An Investigation of the Laws of Thought, on Which are Founded the Mathematical Theories of Logic and Probabilities*. Walton and Maberly.

Borrebæk, M. (2014). *ELF WP2 – Modelling guidelines, version 1.15*. ELF Project, URL: http://elfproject.eu/documentation/specification (visited on 12/09/2015).

Breutmann, B., Falkenberg, E. and Mauer, R. (1979). 'CSL: A Language for Defining Conceptual Schemas'. In: *Data Base Architecture*. Ed. by Bracchi, G. and Nijssen, G. M. North Holland Publishing Company, pp. 337–356.

Carlson, D. (2008). *UML Profile for XML Schema*. Report. URL: http://xmlmodeling.com/specifications/.

Chen, P. P.-S. (1976). 'The Entity-relationship Model—Toward a Unified View of Data'. In: *ACM Trans. Database Syst.* 1.1, pp. 9–36.

Codd, E. F. (1970). 'A Relational Model of Data for Large Shared Data Banks'. In: *Commun. ACM* 13.6, pp. 377–387.

Czarnecki, K. and Helsen, S. (2006). 'Feature-based survey of model transformation approaches'. In: *IBM Systems Journal* 45.3, pp. 621–645.

Data Harmonisation Panel (2012). *User documentation for the CST Web Processing Service*. URL: http://www.esdi-community.eu/projects/hale/wiki/User_documentation_for_the_CST_Web_%20Processing_Service (visited on 13/08/2015).

Data Harmonisation Panel (2015a). *HALE User Guide*. URL: http://hale.igd.fraunhofer.de/2.9.3/help/index.jsp (visited on 13/08/2015).

Data Harmonisation Panel (2015b). *HUMBOLDT Alignment Editor*. URL: http://www.esdi-community.eu/projects/hale (visited on 13/08/2015).

De Morgan, A. (1847). *Formal logic: or, The Calculus of Inference, Necessary and Probable*. Taylor and Walton.

deegree.org (2015). *deegree*. URL: http://www.deegree.org/ (visited on 09/10/2015).

Devillers, R. and Jeansoulin, R. (2006). 'Spatial Data Quality: Concepts'. In: *Fundamentals of Spatial Data Quality*. ISTE, pp. 31–42.

Donaubauer, A., Kutzner, T., Gnägi, H. R., Henrich, S. and Fichtinger, A. (2010). 'Webbasierte Modelltransformation in der Geoinformatik'. In: *Modellierung 2010. 24. - 26. März 2010, Klagenfurt, Österreich*. Ed. by Engels, G., Karagiannis, D. and Mayr, H. Vol. P-161. Lecture Notes in Informatics - Proceedings. Gesellschaft für Informatik, pp. 269–284.

Donaubauer, A., Staub, P., Straub, F. and Fichtinger, A. (2008). 'Web-basierte Modelltransformation – eine Lösung für INSPIRE?' In: *GIS - Zeitschrift für Geoinformatik* 2, pp. 26–33.

Eessaar, E. (2008). 'On Translation-Based Design of UML Profiles'. In: *Innovative Techniques in Instruction Technology, E-learning, E-assessment, and Education*. Ed. by Iskander, M. Springer Netherlands, pp. 144–149.

Egenhofer, M. J. (1993). 'What's Special about Spatial? Database Requirements for Vehicle Navigation in Geographic Space'. In: *Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data, Washington, D.C., May 26-28, 1993*. Ed. by Buneman, P. and Jajodia, S. Vol. 22. SIGMOD Record, pp. 398–402.

Einspanier, U. (2005). 'Formal Metamodeling for the Specification and Implementation of Feature Catalogues'. Dissertation. Universität Münster.

Eisenhut Informatik AG (2006). *ili2fme - INTERLIS-plugin for FME*. URL: http://www.eisenhut%20informatik.ch/interlis/ili2fme/ (visited on 13/08/2015).

Eisenhut, C., Germann, M. and Staub, P. (2011). *GML-Kodierungsregeln für INTERLIS*. eCH-0118. Verein eCH.

Eisenhut, C., Illert, A., Kutzner, T., Müller, M. and Schilcher, M. (2010). *Semantische Datenmodelltransformation im Kontext von INSPIRE - Ergebnisse vom Expertenworkshop des Runder Tisch GIS e.V. am 09.03.2010 an der TU München*. Report. URL: http://www.rtg.bv.tum.de/images/stories/downloads/aus-und_weiterbildung/fortbildungsseminare/2010/WS_SemDat/workshop_semtrans_ergebnissearbeitsgruppen.pdf.

Elaasar, M. and Labiche, Y. (2012). 'Model Interchange Testing: A Process and a Case Study'. In: *Modelling Foundations and Applications. 8th European Conference, ECMFA 2012, Kgs. Lyngby, Denmark, July 2-5, 2012. Proceedings*. Ed. by Vallecillo, A., Tolvanen, J.-P., Kindler, E., Störrle, H. and Kolovos, D. Vol. 7349. Lecture Notes in Computer Science. Berlin Heidelberg: Springer, pp. 49–61.

ELF Project (2015). *European Location Framework*. URL: http://www.elfproject.eu/ (visited on 12/09/2015).

Elfouly, M. and Kutzner, T. (2013). *UMLT Editor Handbook*. Internal report. Technische Universität München, Lehrstuhl für Geoinformatik.

ESRI (1998). *ESRI Shapefile Technical Description*. Environmental Systems Research Institute, Inc.

EuroGeographics (2015). *EuroGeoNames*. URL: http://www.eurogeographics.org/eurogeonames (visited on 22/06/2015).

EuroGeographics (2012). *INSPIRE KEN and EuroSDR workshop « schema transformation »*. URL: http://www.eurogeographics.org/content/inspire-ken-eurosdr-workshop (visited on 13/08/2015).

European Parliament and Council (2007). 'Directive 2007/2/EC of the European Parliament and of the Council of 14 March 2007 establishing an Infrastructure for Spatial Information in the European Community (INSPIRE)'. In: *Official Journal of the European Union* 50.L 108, pp. 1–14.

Euzenat, J. (2015). *EDOAL: Expressive and Declarative Ontology Alignment Language*. URL: http://alignapi.gforge.inria.fr/edoal.html (visited on 22/08/2015).

Fichtinger, A. (2011). 'Semantische Transformation im Kontext von INSPIRE – dargestellt am Beispiel der grenzüberschreitenden Bodenseeregion'. Dissertation. Technische Universität München.

Fitzner, D., Jezek, J., Kolar, J. and Reitz, T. (2009). *A5.2-D3 [3.6] HUMBOLDT Processing Components General Model and Implementations*. Humboldt Consortium.

Friis-Christensen, A., Lutz, M., Cao, H. and Quaglia, A. (2008). *WP3.6 – OA Service Implementation – Implementation Specification of the Translating Feature Access Service*. ORCHESTRA consortium. URL: http://www.eu-orchestra.org/publications.shtml#OAspecs (visited on 19/07/2015).

Fuentes-Fernández, L. and Vallecillo-Moreno, A. (2004). 'An Introduction to UML Profiles'. In: *UPGRADE - The European Journal for the Informatics Professional* V.2, pp. 6–13.

Geonovum (2011). *Implementatiebestand: NEN3610:2011 UML profiel*. URL: http://www.geonovum.nl/documenten/implementatiebestand-nen36102011-uml-profiel (visited on 12/10/2015).

Gröger, G., Kolbe, T. H., Nagel, C. and Häfele, K.-H., eds. (2012). *OGC City Geography Markup Language (CityGML) Encoding Standard, Version 2.0.0*. OGC Document 12-019. Open Geospatial Consortium.

Gruber, T. (1995). 'Toward Principles for the Design of Ontologies Used for Knowledge Sharing'. In: *International Journal Human-Computer Studies* 43.5-6, pp. 907–928.

Häfele, K.-H. (2013). *CityGML-ADEs*. URL: http://www.citygmlwiki.org/index.php/CityGML-ADEs (visited on 24/10/2015).

Henderson-Sellers, B. (2011). 'Random Thoughts on Multi-level Conceptual Modelling'. In: *The Evolution of Conceptual Modeling. From a Historical Perspective towards the Future of Conceptual*

Modeling. Ed. by Kaschek, R. and Delcambre, L. Lecture Notes in Computer Science. Berlin Heidelberg: Springer, pp. 93–116.

Hesse, W. and Mayr, H. C. (2008). 'Modellierung in der Softwaretechnik: eine Bestandsaufnahme'. In: *Informatik-Spektrum* 31.5, pp. 377–393.

Hitz, M., Kappel, G., Kapsammer, E. and Retschitzegger, W. (2005). *UML@Work*. 3., aktualisierte und überarbeitete Auflage. Heidelberg: dpunkt.verlag.

Howard, M., Payne, S. and Sunderland, R. (2010). *Technical Guidance for the INSPIRE Schema Transformation Network Service*. RSW Geomatics, 1Spatial and Rob Walker Consultancy.

Illert, A. and Afflerbach, S. (2003). *Report on Global Schema, GiMoDig Project, Deliverable D5.2.1*. Report. URL: http://gimodig.fgi.fi/.

INTERLIS - The GeoLanguage (2015). URL: http://www.interlis.ch (visited on 09/10/2015).

ISO (1998). *ISO/IEC TR 10000-1:1998 Information technology — Framework and taxonomy of International Standardized Profiles — Part 1: General principles and documentation framework*.

ISO (2004). *ISO 19106:2004 Geographic information — Profiles*.

ISO (2005a). *ISO 19109:2005 Geographic information — Rules for application schema*.

ISO (2005b). *ISO/IEC 19501:2005 Information technology — Open Distributed Processing — Unified Modeling Language (UML) Version 1.4.2*.

ISO (2005c). *ISO/TS 19103:2005 Geographic information — Conceptual schema language*.

ISO (2007). *ISO 19136:2007 Geographic information — Geography Markup Language (GML)*.

ISO (2011). *ISO 19118:2011 Geographic information — Encoding*.

ISO (2012a). *ISO/IEC 19505-1:2012 Information technology — Object Management Group Unified Modeling Language (OMG UML) — Part 1: Infrastructure*.

ISO (2012b). *ISO/IEC 19505-2:2012 Information technology — Object Management Group Unified Modeling Language (OMG UML) — Part 2: Superstructure*.

ISO (2013a). *ISO/DIS 19103:2013 Geographic information — Conceptual schema language*.

ISO (2013b). *ISO/DIS 19109:2013 Geographic information — Rules for application schema*.

ISO (2014). *ISO 19101-1:2014 Geographic information — Reference model — Part 1: Fundamentals*.

Jouault, F. and Kurtev, I. (2005). 'Transforming Models with ATL'. In: *Satellite Events at the MoDELS 2005 Conference*. Ed. by Bruel, J.-M. Lecture Notes in Computer Science. Berlin Heidelberg: Springer, pp. 128–138.

JRC (2012). *UML Profile for INSPIRE data specifications*. URL: http://inspire.jrc.ec.europa.eu/uml/INSPIRE_UMLProfile_20130122.xml (visited on 17/08/2014).

JRC (2013). *D2.8.III.2 Data Specification on Buildings – Technical Guidelines*. European Commission Joint Research Centre.

JRC (2014a). *D2.5: Generic Conceptual Model, Version 3.4*. European Commission Joint Research Centre.

JRC (2014b). *D2.7: Guidelines for the encoding of spatial data, Version 3.3*. European Commission Joint Research Centre.

JRC (2014c). *D2.8.I.3 Data Specification on Geographical Names – Technical Guidelines*. European Commission Joint Research Centre.

JRC (2014d). *D2.8.I.4 Data Specification on Administrative Units – Technical Guidelines*. European Commission Joint Research Centre.

JRC (2014e). *D2.8.I.6 Data Specification on Cadastral Parcels – Technical Guidelines*. European Commission Joint Research Centre.

JRC (2014f). *D2.8.I.9 Data Specification on Protected Sites – Technical Guidelines*. European Commission Joint Research Centre.

JRC (2015). *INSPIRE data models*. URL: http://inspire.ec.europa.eu/index.cfm/pageid/2/list/datamodels (visited on 04/10/2015).

Kay, M., ed. (2007). *XSL Transformations (XSLT) Version 2.0*. W3C. URL: http://www.w3.org/TR/2007/REC-xslt20-20070123/ (visited on 12/08/2015).

Kemper, A. and Eickler, A. (2013). *Datenbanksysteme - Eine Einführung*. Oldenbourg.

Kifer, M. and Boley, H., eds. (2013). *RIF Overview (Second Edition)*. W3C. URL: http://www.w3.org/TR/2013/NOTE-rif-overview-20130205/ (visited on 12/08/2015).

Kleppe, A., Warmer, J. and Bast, W. (2003). *MDA Explained: The Model Driven Architecture: Practice and Promise*. Addison-Wesley Object Technology Series. Boston, MA: Addison-Wesley.

KOGIS, ed. (2006). *INTERLIS Version 2 – Reference Manual*. Edition 2006-04-13. Koordination der Geoinformation und geografischen Informationssysteme (KOGIS).

KOGIS, ed. (2008). *INTERLIS 2 – Metamodel*. Edition 2008-08-29. Koordination der Geoinformation und geografischen Informationssysteme (KOGIS).

KOGIS (2015). *UML/INTERLIS-editor*. URL: http://www.umleditor.org/ (visited on 12/09/2015).

Kolbe, T. H. (2009). 'Representing and Exchanging 3D City Models with CityGML'. In: *3D Geo-Information Sciences*. Ed. by Lee, J. and Zlatanova, S. Lecture Notes in Geoinformation and Cartography. Berlin Heidelberg: Springer, pp. 15–31.

Kottman, C., ed. (1999). *The OpenGIS Abstract Specification, Topic 10: Feature Collections, Version 4*. OGC Document 99-110. Open GIS Consortium.

KPIT medini Technologies (2012). *mediniQVT*. URL: http://projects.ikv.de/qvt/ (visited on 07/06/2015).

Kutzner, T. and Donaubauer, A. (2012). 'Critical Remarks on the Use of Conceptual Schemas in Geospatial Data Modelling - A Schema Translation Perspective'. In: *Bridging the Geographic Information Sciences*. Ed. by Gensel, J., Josselin, D. and Vandenbroucke, D. Lecture Notes in Geoinformation and Cartography. Berlin Heidelberg: Springer, pp. 43–60.

Kutzner, T., Donaubauer, A., Müller, M., Feichtner, A. and Goller, S. (2014). 'Erfolgreiche Transformation von Geodaten nach INSPIRE in der grenzüberschreitenden Region Bodensee'. In: *zfv – Zeitschrift für Geodäsie, Geoinformation und Landmanagement* 2, pp. 103–109.

Kutzner, T. and Eisenhut, C. (2010). *Vergleichende Untersuchungen zur Modellierung und Modelltransformation in der Region Bodensee im Kontext von INSPIRE*. Technische Universität München, Geodätisches Institut.

Kutzner, T. and Kolbe, T. H., eds. (2016). *The OGC CityGML EA UML Model – An ISO-compliant definition of the CityGML 2.0 UML model using Enterprise Architect*. OGC Document. Not yet published. Open Geospatial Consortium.

Kutzner, T., Schilcher, M. and Aderhold, B. (2012). 'INSPIRE auf dem Prüfstand der grenzüberschreitenden Praxistauglichkeit in der Testregion Bodensee'. In: *24. Symposium für Angewandte Geoinformatik (AGIT)*. Ed. by Strobl, J., Blaschke, T. and Griesebner, G. Wichmann, pp. 181–190.

Lagarde, F., Espinoza, H., Terrier, F., André, C. and Gérard, S. (2008). 'Leveraging Patterns on Domain Models to Improve UML Profile Definition'. In: *Fundamental Approaches to Software Engineering. 11th International Conference, FASE 2008, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2008, Budapest, Hungary, March 29-April 6, 2008. Proceedings*. Ed. by Fiadeiro, J. L. and Inverardi, P. Vol. 4961. Lecture Notes in Computer Science. Berlin Heidelberg: Springer, pp. 116–130.

Latvala, P., Lassi, L. and Kähkönen, J. (2013). 'The Renewed Implementation of the EuroGeoNames Central Service'. In: *Geographic Information Science at the Heart of Europe, The 16th AGILE International Conference on Geographic Information Science*. Ed. by Vandenbroucke, D., Bucher,

B. and Crompvoets, J. URL: https://agile-online.org/Conference_Paper/CDs/agile_2013/Posters/P_Latvala.pdf.

Lehto, L. (2007a). 'Real-time content transformations in a web service-based delivery architecture for geographic information'. Dissertation. Helsinki University of Technology.

Lehto, L. (2007b). 'Schema Translations in a Web Service Based SDI'. In: *10th AGILE International Conference on Geographic Information Science*. URL: http://www.agile-online.org/Conference_Paper/CDs/agile_2007/PROC/PDF/29_PDF.pdf.

Lehto, L., ed. (2011). *ESDIN – Recommendations for Operational Deployment of Services*. EuroGeographics. URL: http://www.esdin.eu/project/summary-esdin-project-public-deliverables (visited on 20/07/2015).

Lehto, L. and Sarjakoski, T. (2004). 'Schema translations by XSLT for GML-encoded geospatial data in heterogeneous Webservice environment'. In: *XXth ISPRS Congress*. Vol. XXXV. The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences. International Society of Photogrammetry and Remote Sensing (ISPRS), pp. 177–182.

Leser, U. and Naumann, F. (2007). *Informationsintegration*. dpunkt.verlag.

Lill, M., Kutzner, T. and Fünfer, H. (2011). 'Experience with Semantic Transformation Based on Currently Available Software'. In: *INSPIRE-GMES Information Brochure*. Ed. by Schilcher, M. 7th completely revised, extended and updated edition. Technische Universität München, Geodätisches Institut, pp. 49–51.

Lisboa-Filho, J., Nalon, F. R., Peixoto, D. A., Sampaio, G. B. and Vasconcelos Borges, K. A. de (2013). 'Domain and Model Driven Geographic Database Design'. In: *Domain Engineering*. Ed. by Reinhartz-Berger, I., Sturm, A., Clark, T., Cohen, S. and Bettin, J. Berlin Heidelberg: Springer, pp. 375–399.

Lisboa-Filho, J., Sampaio, G. B., Nalon, F. R. and Vasconcelos Borges, K. A. de (2010). 'A UML Profile for Conceptual Modeling in GIS Domain'. In: *Proceedings of the International Workshop on Domain Engineering (DE@CAiSE 2010), Hammamet, Tunisia, June 8, 2010*. Ed. by Reinhartz-Berger, I., Sturm, A., Wand, Y., Bettin, J., Clark, T., Cohen, S., Ralyté, J. and Plebani, P. Vol. Vol-602. CEUR-WS.org, pp. 18–31.

Löwner, M.-O., Benner, J. and Gröger, G. (2014). 'Aktuelle Trends in der Entwicklung von CityGML 3.0'. In: *Geoinformationen öffnen das Tor zur Welt, 34. Wissenschaftlich-Technische Jahrestagung der DGPF in Hamburg*. Ed. by Seyfert, E., Gülch, E., Heipke, C., Schiewe, J. and Sester, M. Vol. 23. Publikationen der Deutschen Gesellschaft für Photogrammetrie, Fernerkundung, Geoinformation e.V. Deutsche Gesellschaft für Photogrammetrie, Fernerkundung, Geoinformation e.V.

Lundell, B., Lings, B., Persson, A. and Mattsson, A. (2006). 'UML Model Interchange in Heterogeneous Tool Environments: An Analysis of Adoptions of XMI 2'. In: *Model Driven Engineering Languages and Systems. 9th International Conference, MoDELS 2006, Genova, Italy, October 1-6, 2006. Proceedings*. Ed. by Nierstrasz, O., Whittle, J., Harel, D. and Reggio, G. Vol. 4199. Lecture Notes in Computer Science. Berlin Heidelberg: Springer, pp. 619–630.

Mens, T. and Gorp, P. van (2006). 'A Taxonomy of Model Transformation'. In: *Electronic Notes in Theoretical Computer Science* 152, pp. 125–142.

Nagel, C., Stadler, A. and Kolbe, T. H. (2009). 'Conceptual Requirements for the Automatic Reconstruction of Building Information Models from Uninterpreted 3D Models'. In: *Proceedings of the GeoWeb 2009 Academic Track - Cityscapes, Vancouver,Canada, 27-31 July 2009*. Ed. by Kolbe, T. H., Zhang, R. and Zlatanova, S. Vol. XXXVIII-3-4/C3. The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences. ISPRS, pp. 46–53.

Negri, M. and Pelagatti, G. (2014). *GeoUML Tools*. URL: http://geo.spatialdbgroup.polimi.it/en/geouml-tools/ (visited on 25/08/2015).

Nissen, F., Friis-Christensen, A., Münster-Svendsen, J., Rykov, J. B. and Nielsen, Å. (2011). *ESDIN – D10.2 Framework for Specifying Transformation Rules*. EuroGeographics. URL: http://www.esdin.eu/project/summary-esdin-project-public-deliverables (visited on 20/07/2015).

No Magic (2015). *MagicDraw*. URL: http://www.nomagic.com/products/magicdraw.html (visited on 04/10/2015).

Noyrit, F., Gérard, S., Terrier, F. and Selic, B. (2010). 'Consistent Modeling Using Multiple UML Profiles'. In: *Model Driven Engineering Languages and Systems. 13th International Conference, MODELS 2010, Oslo, Norway, October 3-8, 2010, Proceedings, Part I*. Ed. by Petriu, D. C., Rouquette, N. and Haugen, Ø. Vol. 6394. Lecture Notes in Computer Science. Berlin Heidelberg: Springer, pp. 392–406.

Obrst, L. (2003). 'Ontologies for semantically interoperable systems'. In: *Conference on Information and Knowledge Management (CIKM)*. ACM New York, NY, USA, pp. 366–369.

OMG (2002). *Request for Proposal: MOF 2.0 Query / Views / Transformations RFP*. OMG Document ad/2002-04-10.

OMG (2008). *MOF Model to Text Transformation Language, v1.0*. OMG Document formal/2008-01-16. URL: http://www.omg.org/spec/MOFM2T/1.0/PDF.

OMG (2011a). *Meta Object Facility (MOF) 2.0 Query/View/Transformation Specification Version 1.1*. OMG Document formal/2011-01-01.

OMG (2011b). *OMG Unified Modeling Language (OMG UML), Infrastructure Version 2.4.1*. OMG Document formal/2011-08-05.

OMG (2011c). *OMG Unified Modeling Language (OMG UML), Superstructure Version 2.4.1*. OMG Document formal/2011-08-06.

OMG (2012). *OMG Object Constraint Language (OCL) Version 2.3.1*. OMG Document formal/2012-01-01.

OMG (2014a). *MDA Guide Revision 2.0*. OMG Document ormsc/2014-06-01.

OMG (2014b). *OMG Meta Object Facility (MOF) Core Specification Version 2.4.2*. OMG Document formal/2014-04-03.

OMG (2014c). *XML Metadata Interchange (XMI) Specification Version 2.4.2*. OMG Document formal/2014-04-04.

OMG (2015). *OMG Systems Modeling Language (OMG SysML) Version 1.4*. OMG Document formal/2015-06-03.

Oracle (2015). *Oracle Spatial and Graph*. URL: http://www.oracle.com/us/products/database/options/spatial/overview/index.html (visited on 09/10/2015).

Pardillo, J. (2010). 'A Systematic Review on the Definition of UML Profiles'. In: *Model Driven Engineering Languages and Systems. 13th International Conference, MODELS 2010, Oslo, Norway, October 3-8, 2010, Proceedings, Part I*. Ed. by Petriu, D. C., Rouquette, N. and Haugen, Ø. Vol. 6394. Lecture Notes in Computer Science. Berlin Heidelberg: Springer, pp. 407–422.

Portele, C. (2012a). 'Encoding of Geographic Information'. In: *Springer Handbook of Geographic Information*. Ed. by Kresse, W. and Danko, D. M. Berlin Heidelberg: Springer, pp. 35–47.

Portele, C., ed. (2012b). *OGC Geography Markup Language (GML) — Extended schemas and encoding rules, Version 3.3.0*. OGC Document 10-129r1. Open Geospatial Consortium.

Portele, C., Erstling, R. and Olk, S. (2014). *AFIS-ALKIS-ATKIS-Tools auf der Basis von ShapeChange – Documentation*. Report. interactive instruments GmbH.

Reitz, T., Schäffler, U., Klien, E. and Fitzner, D. (2010). 'Efficient Conceptual Schema Translation for Geographic Vector Data Sets'. In: *Proceedings of the AGILE'2010 International Conference on Geographic Information Science, Guimarães, 11-14, May, 2010*. Ed. by Painho, M., Santos, M. Y. and Pundt, H. URL: http://plone.itc.nl/agile_old/Conference/2010-guimaraes/shortPapers.html.

Reitz, T. and Templer, S. (2012). 'An Environment for the Conceptual Harmonisation of Geospatial Schemas and Data'. In: *Proceedings of the AGILE'2012 International Conference on Geographic Information Science, Avignon, April, 24-27, 2012*. Ed. by Gensel, J., Josselin, D. and Vandenbroucke, D., pp. 63–68.

Reitz, T., Vries, M. de and Fitzner, D. (2009). *A5.2-D3 [3.3] Conceptual Schema Specification and Mapping*. Humboldt Consortium.

Riccobene, E. and Scandurra, P. (2012). 'Integrating the SysML and the SystemC-UML profiles in a model-driven embedded system design flow'. In: *Design Automation for Embedded Systems* 16.3, pp. 53–91.

Roswell, C. (2012). 'Modeling of Geographic Information'. In: *Springer Handbook of Geographic Information*. Ed. by Kresse, W. and Danko, D. M. Berlin Heidelberg: Springer, pp. 3–17.

Safe Software (2005). *Building Applications with FME Objects*. Report. URL: ftp://ftp.safe.com/fme/fme_objects/.

Safe Software (2015a). *FME Desktop*. URL: http://www.safe.com/fme/fme-desktop/ (visited on 13/08/2015).

Safe Software (2015b). *FME Factory and Function Documentation*. URL: http://docs.safe.com/fme/html/FME_FactFunc/index.html (visited on 30/08/2015).

Safe Software (2015c). *Spatial ETL*. URL: http://www.safe.com/fme/key-capabilities/spatial-etl/ (visited on 10/10/2015).

Schönherr, H., Schilcher, M., Illert, A., Kutzner, T. and Heß, D. (2011). 'Cross-border Transformation of Spatial Data to INSPIRE in the Lake Constance Region – Challenge for Administration, Industry and Science'. In: *INSPIRE-GMES Information Brochure*. Ed. by Schilcher, M. 7th completely revised, extended and updated edition. Technische Universität München, Geodätisches Institut, pp. 44–48.

Schulze Althoff, J. (2011). 'Model-driven tools to support conceptual geospatial modelling'. Dissertation. ETH Zürich.

SEE Grid (2009). *UML Profile for GML Applications Schemas*. URL: https://www.seegrid.csiro.au/subversion/HollowWorld/branches/release_4/Utilities/ISO19136_UMLProfile.xml (visited on 17/08/2014).

SEE Grid (2010). *Overview of some relevant standards from ISO/TC 211*. URL: https://www.seegrid.csiro.au/wiki/AppSchemas/IsoTc211Standards (visited on 04/10/2014).

Selic, B. (2007). 'A Systematic Approach to Domain-Specific Language Design Using UML'. In: *Proceedings of the 10th IEEE International Symposium on Object and Component-Oriented Real-Time Distributed Computing (ISORC'07)*. IEEE, pp. 2–9.

Selic, B. (2012). 'The Less Well Known UML'. In: *Formal Methods for Model-Driven Engineering. 12th International School on Formal Methods for the Design of Computer, Communication, and Software Systems, SFM 2012, Bertinoro, Italy, June 18-23, 2012. Advanced Lectures*. Ed. by Bernardo, M., Cortellessa, V. and Pierantonio, A. Vol. 7320. Lecture Notes in Computer Science. Berlin Heidelberg: Springer, pp. 1–20.

Sheth, A. P. (1999). 'Changing Focus on Interoperability in Information Systems: From System, Syntax, Structure to Semantics'. In: *Interoperating Geographic Information Systems*. Ed. by

Goodchild, M., Egenhofer, M., Fegeas, R. and Kottman, C. Vol. 495. The Springer International Series in Engineering and Computer Science. Springer US. Chap. 2, pp. 5–29.

Snowflake Software (2014). *GO Publisher*. URL: http://www.snowflakesoftware.com/products/gopublisher/ (visited on 13/08/2015).

Sparx Systems (2015). *Enterprise Architect*. URL: http://www.sparxsystems.com/products/ea/index.html (visited on 04/10/2015).

Spyns, P., Meersman, R. and Jarrar, M. (2002). 'Data modelling versus ontology engineering'. In: *ACM SIGMOD Record* 31.4, pp. 12–17.

Stachowiak, H. (1973). *Allgemeine Modelltheorie*. Springer Vienna.

Stahl, T. and Völter, M. (2006). *Model-Driven Software Development: Technology, Engineering, Management*. John Wiley & Sons.

Staron, M. and Kuzniarz, L. (2005). 'Properties of Stereotypes from the Perspective of Their Role in Designs'. In: *Model Driven Engineering Languages and Systems. 8th International Conference, MoDELS 2005, Montego Bay, Jamaica, October 2-7, 2005. Proceedings*. Ed. by Briand, L. and Williams, C. Vol. 3713. Lecture Notes in Computer Science. Berlin Heidelberg: Springer, pp. 201–216.

Staub, P. (2007). 'A Model-Driven Web Feature Service for Enhanced Semantic Interoperability'. In: *OSGeo Journal* 3, pp. 38–43.

Staub, P. (2009). 'Über das Potenzial und die Grenzen der semantischen Interoperabilität von Geodaten: Ein operationelles Verfahren zur Nutzung verteilter Systeme in Geodaten-Infrastrukturen'. Dissertation. ETH Zürich.

Staub, P., Gnägi, H. R. and Morf, A. (2008). 'Semantic Interoperability through the Definition of Conceptual Model Transformations'. In: *Transactions in GIS* 12.2, pp. 193–207.

Steel Jr., T. B. (1975). 'Data Base Standardization: A Status Report'. In: *International Conference on Management of Data (MOD)*. SIGMOD '75. ACM, pp. 149–156.

Steinberg, D., Budinsky, F., Paternostro, M. and Merks, E. (2009). *EMF: Eclipse Modeling Framework*. The Eclipse Series. Addison-Wesley Professional.

The Eclipse Foundation (2015a). *ATL*. URL: http://www.eclipse.org/atl/ (visited on 18/05/2015).

The Eclipse Foundation (2015b). *ATL/User Guide - The ATL Language*. URL: http://wiki.eclipse.org/ATL/User_Guide_-_The_ATL_Language (visited on 18/05/2015).

The Eclipse Foundation (2015c). *Eclipse Modeling Framework (EMF)*. URL: https://eclipse.org/modeling/emf/ (visited on 29/08/2015).

The Eclipse Foundation (2015d). *Graphical Modeling Project (GMP)*. URL: http://www.eclipse.org/modeling/gmp/ (visited on 29/08/2015).

The Eclipse Foundation (2015e). *Model Development Tools (MDT) - UML2*. URL: http://www.eclipse.org/modeling/mdt/?project=uml2 (visited on 29/08/2015).

The Eclipse Foundation (2015f). *QVTo*. URL: https://wiki.eclipse.org/QVTo (visited on 07/06/2015).

Usländer, T., ed. (2007). *Reference Model for the ORCHESTRA Architecture (RM-OA) V2 (Rev 2.1)*. OGC Document 07-097. Open Geospatial Consortium.

van den Brink, L., Stoter, J. and Zlatanova, S., eds. (2012). *Modeling an application domain extension of CityGML in UML*. OGC document 12-066. Open Geospatial Consortium.

Wagelaar, D. (2008). 'Composition Techniques for Rule-Based Model Transformation Languages'. In: *Theory and Practice of Model Transformations*. Ed. by Vallecillo, A., Gray, J. and Pierantonio, A. Vol. 5063. Lecture Notes in Computer Science. Berlin Heidelberg: Springer, pp. 152–167.

Wagelaar, D. (2010). *Log of /UML2CaseStudies/uml2cs-transformations/transformations/UML2Copy.atl*. URL: http://soft.vub.ac.be/viewvc/UML2CaseStudies/uml2cs-transformations/transformations/UML2Copy.atl (visited on 02/08/2015).

Warmer, J. and Kleppe, A. (2003). *The Object Constraint Language, Second Edition: Getting Your Models Ready for MDA*. 2nd. Addison-Wesley Object Technology Series. Boston, MA: Addison-Wesley.

Worboys, M. and Duckham, M. (2004). *GIS: A Computing Perspective, Second Edition*. Boca Raton: CRC Press.

Original publications

The author of this thesis was involved in the publication of several papers and articles written in the context of those research projects, the results of this thesis are based on and which are mentioned in the introduction chapter, some of these papers and articles describing individual aspects which are also addressed in this work.

- Kutzner, T., Donaubauer, A., Müller, M., Feichtner, A. and Goller, S. (2014). 'Erfolgreiche Transformation von Geodaten nach INSPIRE in der grenzüberschreitenden Region Bodensee'. In: zfv – Zeitschrift für Geodäsie, Geoinformation und Landmanagement 2, pp. 103–109.
- Kutzner, T. and Donaubauer, A. (2012). 'Critical Remarks on the Use of Conceptual Schemas in Geospatial Data Modelling – A Schema Translation Perspective'. In: Bridging the Geographic Information Sciences. Ed. by Gensel, J., Josselin, D. and Vandenbroucke, D. Lecture Notes in Geoinformation and Cartography. Berlin Heidelberg: Springer, pp. 43–60.
- Kutzner, T. (2012). 'Vergleichende Untersuchungen zur Modellierung und semantischen Modelltransformation in der Region Bodensee'. In: Geoinformationssysteme, Beiträge zum 17. Münchner Fortbildungsseminar 2012. Ed. by Koch, A.,Kutzner, T. and Eder, T. Wichmann, pp. 61–72.
- Kutzner, T., Schilcher, M. and Aderhold, B. (2012). 'INSPIRE auf dem Prüfstand der grenzüberschreitenden Praxistauglichkeit in der Testregion Bodensee'. In: 24. Symposium für Angewandte Geoinformatik (AGIT). Ed. by Strobl, J., Blaschke, T. and Griesebner, G. Wichmann, pp. 181–190.
- Donaubauer, A., Kutzner, T. and Straub, F. (2011). 'Model-driven Semantic Transformation – a Report on Work in Progress'. In: INSPIRE-GMES Information Brochure. Ed. by Schilcher, M. 7th completely revised, extended and updated edition. Technische Universität München, Geodätisches Institut, pp. 52–53.
- Donaubauer, A., Kutzner, T. und Straub, F. (2011). 'Modellbasierte semantische Transformation – ein Werkstattbericht'. In: INSPIRE-GMES-Informationsbroschüre. Ed. by Schilcher, M. 7. vollständig überarbeitete und aktualisierte Auflage. Technische Universität München, Geodätisches Institut, pp. 60–62.
- Schönherr, H., Schilcher, M., Illert, A., Kutzner, T. and Heß, D. (2011). 'Cross-border Transformation of Spatial Data to INSPIRE in the Lake Constance Region – Challenge for Administration, Industry and Science'. In: INSPIRE-GMES Information Brochure. Ed. by Schilcher, M. 7th completely revised, extended and updated edition. Technische Universität München, Geodätisches Institut, pp. 44–48.
- Schönherr, H., Schilcher, M., Illert, A., Kutzner, T. und Heß, D. (2011). 'Grenzüberschreitende Transformation von Geodaten nach INSPIRE in der Region Bodensee – Herausforderung für Verwaltung, Wirtschaft und Wissenschaft'. In: INSPIRE-GMES-Informationsbroschüre. Ed. by Schilcher, M. 7. vollständig überarbeitete und aktualisierte Auflage. Technische Universität München, Geodätisches Institut, pp. 51–55.

- Lill, M., Kutzner, T. and Fünfer, H. (2011). 'Experience with Semantic Transformation Based on Currently Available Software'. In: INSPIRE-GMES Information Brochure. Ed. by Schilcher, M. 7th completely revised, extended and updated edition. Technische Universität München, Geodätisches Institut, pp. 49–51.
- Lill, M., Kutzner, T. und Fünfer, H. (2011). 'Erfahrungen mit semantischer Transformation auf Basis heute verfügbarer Software'. In: INSPIRE-GMES-Informationsbroschüre. Ed. by Schilcher, M. 7. vollständig überarbeitete und aktualisierte Auflage. Technische Universität München, Geodätisches Institut, pp. 56–59.
- Kutzner, T. and Eisenhut, C. (2010). Vergleichende Untersuchungen zur Modellierung und Modelltransformation in der Region Bodensee im Kontext von INSPIRE. Technische Universität München, Geodätisches Institut.
- Donaubauer, A., Kutzner, T., Gnägi, H. R., Henrich, S. and Fichtinger, A. (2010). 'Webbasierte Modelltransformation in der Geoinformatik'. In: Modellierung 2010. Ed. by Engels, G., Karagiannis, D. and Mayr, H. Vol. P-161. Lecture Notes in Informatics - Proceedings. Bonn: Gesellschaft für Informatik, pp. 269–284.
- Kutzner, T. (2010). 'The Unified Modeling Language for the Translation of Spatial Data on Conceptual Level – Challenges and Limitations'. In: Proceedings of the GIScience 2010 Doctoral Colloquium. Ed. by Wallgrün, J. O. and Lautenschütz, A.-K. ifgiPrints. Akademische Verlagsgesellschaft AKA, pp. 55–58.
- Fichtinger, A. and Kutzner, T. (2010). 'Datenharmonisierung im Kontext von INSPIRE'. In: Geoinformationssysteme – Beiträge zum 15. Münchner Fortbildungsseminar. Ed. by Schilcher, M. abcverlag, pp. 30–46.
- Eisenhut, C. and Kutzner, T. (2010). 'Vergleichende Untersuchungen zur Modellierung und Modelltransformation in der Region Bodensee im Kontext von INSPIRE'. In: INSPIRE-GMES-Informationsbroschüre. Ed. by Schilcher, M. 6. vollständig überarbeitete und aktualisierte Auflage. Technische Universität München, Geodätisches Institut, p. 34.
- Kutzner, T. (2010). 'Semantische Transformation am Beispiel der Testregion Bodensee'. In: INSPIRE-GMES-Informationsbroschüre. Ed. by Schilcher, M. 6. vollständig überarbeitete und aktualisierte Auflage. Technische Universität München, Geodätisches Institut, pp. 35–36.
- Banfi, D., Fünfer, H. and Kutzner, T. (2010). 'Von ALKIS und ATKIS zu INSPIRE'. In: INSPIRE-GMES-Informationsbroschüre. Ed. by Schilcher, M. 6. vollständig überarbeitete und aktualisierte Auflage. Technische Universität München, Geodätisches Institut, pp. 40–42.
- Donaubauer, A., Fichtinger, A. and Kutzner, T. (2009). 'Semantische Transformation als Geo Web Service'. In: Mitteilungen des Bundesamtes für Kartographie und Geodäsie. Verlag des Bundesamtes für Kartographie und Geodäsie, pp. 67–82.
- Donaubauer, A., Fichtinger, A., Kutzner, T. and Schilcher, M. (2009). 'Semantische Modelltransformation im Kontext von INSPIRE'. In: Newsletter e-geo.ch. swisstopo, pp. 10–13.