

SEMANTIC VALIDATION IN SPATIO-TEMPORAL SCHEMA INTEGRATION

THÈSE N° 3423 (2006)

PRÉSENTÉE À LA FACULTÉ INFORMATIQUE ET COMMUNICATIONS

Institut d'informatique fondamentale

SECTION D'INFORMATIQUE

ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

POUR L'OBTENTION DU GRADE DE DOCTEUR ÈS SCIENCES

PAR

Anastasiya SOTNYKOVA

M.Sc. in Computer Engineering and Information Science, Bilkent University, Turquie
et de nationalité ukrainienne

acceptée sur proposition du jury:

Prof. S. Spaccapietra, directeur de thèse
Prof. A. Artale, rapporteur
Prof. N. Cullot, rapporteur
Prof. A. Wegmann, rapporteur

Lausanne, EPFL
2006

Abstract

Semantic Validation in Spatio-Temporal Schema Integration

This thesis proposes to address the well-know database integration problem with a new method that combines functionality from database conceptual modeling techniques with functionality from logic-based reasoners. We elaborate on a hybrid - *modeling+validation* - integration approach for spatio-temporal information integration on the schema level. The modeling part of our methodology is supported by the spatio-temporal conceptual model MADS, whereas the validation part of the integration process is delegated to the description logics validation services. We therefore adhere to the principle that, rather than extending either formalism to try to cover all desirable functionality, a hybrid system, where the database component and the logic component would cooperate, each one performing the tasks for which it is best suited, is a viable solution for semantically rich information management.

First, we develop a MADS-based flexible integration approach where the integrated schema designer has several viable ways to construct a final integrated schema. For different related schema elements we provide the designer with four general policies and with a set of structural solutions or structural patterns within each policy. To always guarantee an integrated solution, we provide for a preservation policy with multi-representation structural pattern. To state the inter-schema mappings, we elaborate on a correspondence language with explicit spatial and temporal operators. Thus, our correspondence language has three facets: structural, spatial, and temporal, allowing to relate the thematic representation as well as the spatial and temporal features. With the inter-schema mappings, the designer can state correspondences between related populations, and define the conditions that rule the matching at the instance level. These matching rules can then be used in query rewriting procedures or to match the instances within the data integration process. We associate a set of putative structural patterns to each type of population correspondence, providing a designer with a patterns' selection for flexible integrated schema construction.

Second, we enhance our integration method by employing validation services of the description logic formalism. It is not guaranteed that the designer can state all the inter-schema mappings manually, and that they are all correct. We add the validation phase to ensure validity and completeness of the inter-schema mappings set. Inter-schema mappings cannot be validated autonomously, i.e., they are validated against the data model and the schemas they link. Thus, to implement our validation approach, we translate the data model, the source schemas and the inter-schema mappings into a description logic formalism, preserving the spatial and temporal semantics of the MADS data model. Thus, our modeling approach in description logic insures that the model designer will correctly define spatial and

temporal schema elements and inter-schema mappings. The added value of the complete translation (i.e., including the data model and the source schemas) is that we validate not only the inter-schema mappings, but also the compliance of the source schemas to the data model, and infer implicit relationships within them. As the result of the validation procedure, the schema designer obtains the complete and valid set of inter-schema mappings and a set of valid (flexible) schematic patterns to apply to construct an integrated schema that meets application requirements.

To further our work, we model a framework in which a schema designer is able to follow our integration method and realize the schema integration task in an assisted way. We design two models, UML and SEAM models, of a system that provides for integration functionalities. The models describe a framework where several tools are employed together, each involved in the service it is best suited for. We define the functionalities and the cooperation between the composing elements of the framework and detail the logics of the integration process in an UML activity diagram and in a SEAM operation model.

Version Abrégée

Validation Sémantique pour l'Intégration de Schémas Spatio-Temporels.

Ce travail de recherche aborde la problématique d'intégration de base de données et propose une nouvelle méthode qui allie les techniques de modélisation conceptuelle de bases de données avec les capacités de raisonnement des logiques de description. Nous avons élaboré une approche hybride - *modélisation+validation* - pour l'intégration de données spatio-temporelles au niveau du schéma. La partie modélisation de notre méthodologie est réalisée avec le modèle conceptuel pour données spatio-temporelles, MADS, et la partie de validation du processus d'intégration est déléguée aux services de raisonnement des logiques de description. En effet, plutôt que d'étendre l'un ou l'autre formalisme et essayer d'offrir toutes les fonctionnalités souhaitées, nous considérons que un système hybride où le composant base de données et le composant logique coopèrent, chacun accomplissant les tâches pour lesquelles il est le plus adapté, est la solution la plus appropriée pour la gestion sémantique de l'information.

Nous avons développé une approche flexible d'intégration où le concepteur du schéma intégré dispose de plusieurs manières valides pour construire un schéma intégré final. Pour chacun des éléments des schémas en correspondance, nous proposons ainsi au concepteur quatre politiques générales d'intégration avec, pour chacune, un ensemble de solutions structurelles (ou de patterns structuraux). Afin de pouvoir toujours offrir au concepteur une solution pour l'intégration de ses schémas, nous avons prévu une politique particulière appelée politique de conservation reposant sur le pattern de multi-représentation. Pour formuler les mappings inter-schémas, nous avons défini un langage de correspondances incluant des opérateurs spatiaux et temporels. Ainsi, notre langage de correspondances possédant trois facettes: structurale, spatiale, et temporelle, il permet de relier non seulement la représentation thématique des données mais aussi leurs propriétés spatiales et temporelles. A l'aide des mappings inter-schémas, le concepteur peut spécifier les correspondances entre les populations reliées, et définir les conditions qui régissent la mise en correspondance des données au niveau instance. Ces règles peuvent alors être employées lors des procédures de re-écriture de requêtes et pour apparier les instances lors du processus d'intégration de données. Nous avons associé à chaque type de correspondance entre des populations un ensemble de solutions structurelles putatives, fournissant au concepteur une sélection de patterns pour la construction d'un schéma intégré flexible.

Nous avons considéré que, utilisant un modèle conceptuel spatio-temporel expressif, le concepteur de schéma est capable de spécifier ses mappings inter-schéma manuellement. Cependant il n'était pas acceptable de considérer que le concepteur

serait capable d'énoncer tous les mappings, et que ces mappings seraient tous corrects. Pour pallier ce problème, nous avons adjoint notre méthode d'intégration une étape de validation permettant d'assurer la validité et l'intégrité de l'ensemble des mappings inter-schémas. Les mappings inter-schémas ne peuvent pas être validés de façon autonome, ils sont validés relativement au modèle de données utilisé et aux schémas qu'ils mettent en correspondance. Ainsi, lors de la validation, nous traduisons tout d'abord le modèle de données MADS, puis les schémas sources et les mappings inter-schémas en une logique de description préservant la sémantique spatiale et temporelle du modèle MADS. Cette étape de validation permet de certifier que le concepteur du schéma a défini correctement les éléments spatiaux et temporels du schéma ainsi que les mappings inter-schémas. La valeur ajoutée de la traduction complète (c.-à-d., incluant le modèle de données et les schémas sources) est que nous validons non seulement les mappings inter-schémas, mais également la conformité des schémas sources au modèle de données, et que cela permet d'inférer des liens implicites entre les schémas sources. A l'issue de la validation, le concepteur de schéma obtient un ensemble complet et valide de mappings inter-schémas, et un ensemble de patterns schématiques (flexibles) valides qu'il peut ensuite appliquer pour construire le schéma intégré répondant aux exigences de son application.

Finalement, nous avons aussi proposé un modèle décrivant une architecture dans laquelle le concepteur de schéma peut appliquer notre méthode d'intégration et être assisté lors de l'intégration de ses schémas sources. Nous avons réalisé le modèle de cette architecture à l'aide de deux formalismes différents, le formalisme UML et le formalisme SEAM. Ces modèles décrivent une architecture dans laquelle plusieurs outils sont utilisés conjointement, chacun étant employé pour le service pour lequel il est le plus adapté. Enfin, nous avons défini les fonctionnalités entre les éléments composants de l'architecture, comment ils coopèrent, et nous avons détaillé la logique du procédé d'intégration dans un diagramme d'activité UML et dans un diagramme d'opération de SEAM.

Acknowledgements

This thesis sums up my research efforts as a member of EPFL's database laboratory (LBD), led by Professor Stefano Spaccapietra who was my advisor throughout my doctoral student years.

It takes many months, marked by endless doubts and the occasional finding, before a PhD thesis attains its final shape. The proverbial Ariadne's thread for me was my advisor's critical eye; it was him who would emphatically ask the very same final question at all intermediate presentations: "What are the contributions of what you just presented?". And, by "contributions", he would mean something new and original within a research topic which had been thoroughly studied for a few decades already. Professor Christine Parent possessed the uncanny ability to point out even the slightest mismatch between my argumentations and other people's. Thanks to her input, I was able to change my vantage point on the subject and see that actually the semantics of boundaries, objects, properties, relationships, and of other phenomena is highly dependent on how we abstract the world.

Working as a member of the team of LBD members, I am thankful to my colleagues who were openly sharing their academic experience and supporting my ideas; they helped me step in a land which is "outside of databases," and the result of that is the present work, which merges two different scientific approaches. To me, LBD was a laboratory with a balanced atmosphere, nourishing the creativity of its members and shaping them personally and professionally.

This work would not have been possible without the infrastructure provided by the Swiss Federal Institute of Technology and the support of the Swiss National Research Foundation. The top-notch school's facilities fostered my research, my teaching and my self-development activities. EPFL provided an ideal framework for me to show and discuss my work in front of a highly competent jury, which provided me with invaluable comments and feedback.

In the "outside" world, my friends provided great support; with them I had many of life's important conversations and they generously shared their passions with me. Being a pathfinder in the labyrinth of formal approaches, I was getting my inspiration from a beautiful mountain view, from a high-speed descent in a white-gleaming valley, from traveling with the fascinating argentinean music far away from the real world, or from eating delicious *haute cuisine* creations. Many thanks to all of you who kept asking me "How is your thesis going on?", and to whom (and therefore to myself!) I ended up explaining the particularities of conceptual modeling and integration. And, of course, my family deserves a special thanks for their faith in me – and my mother for her inexhaustible source of *joie de vivre* which has always been and will be a driving force in many of my endeavors.

Contents

1	Introduction	1
1.1	General Context of the Study	1
1.2	Motivations	4
1.3	Outline of the Thesis	5
2	Relevant Research Areas: State of the Art	7
2.1	Interoperable System Architectures	7
2.1.1	Interoperable System Components	9
2.1.2	Mediator-based systems	10
2.1.3	Agent-based systems	11
2.1.4	Summary	13
2.2	Ontologies and Conceptual Models	15
2.2.1	Ontologies vs Conceptual Schemas	16
2.2.2	The MADS Conceptual Model as a Common Data Model . . .	17
2.3	Inter-Schema Mappings	18
2.3.1	Mapping discovery.	19
2.3.2	Inter-schema correspondences.	19
2.3.3	Querying.	20
2.3.4	Semantic enrichment.	20
2.4	Validation Approach in Integration Procedures	21
2.4.1	Reasoning in Description Logic	22
2.4.2	Description Logics	23
2.4.3	Description Logics with Concrete Domains.	26
2.5	Chapter Summary	30
3	Integration of Spatio-temporal Database Schemas	31
3.1	The Context of the Methodology: Terms and Techniques	32
3.2	Introduction to the MADS data model	34
3.2.1	Structural dimension	34
3.2.2	Spatial dimension	35
3.2.3	Temporal dimension	36
3.2.4	Constrained relationships	37
3.2.5	Multiple Representations with Multiple Perceptions in MADS	39

3.3	Motivating Examples	41
3.4	Integration Methodology	45
3.4.1	Pre-integration	45
3.4.2	Inter-schema Mappings	45
3.4.3	Choosing a structural policy.	59
3.4.4	Integrated schema composition	65
3.5	Chapter Summary	66
4	Validation: Theory	69
4.1	Validation in $\mathcal{ALCRP}(\mathcal{D})$	71
4.1.1	Description Logic $\mathcal{ALCRP}(\mathcal{D})$	71
4.1.2	Spatio-Temporal Concrete Domain	72
4.2	MADS schemas in $\mathcal{ALCRP}(\mathcal{D})$	73
4.2.1	Spatio-temporal features	73
4.2.2	Inter-schema mappings	74
4.2.3	Structural solution for the integrated schema	78
4.2.4	Composing the Integrated Schema	82
4.3	Chapter Summary	84
5	Validation: Practice	85
5.1	MADS data model in \mathcal{SHIQ} logic	88
5.2	Data Model Definition in OWL	93
5.2.1	Structural Dimension.	93
5.2.2	Spatial Dimension.	105
5.2.3	Temporal Dimension.	109
5.2.4	Constrained relationships.	111
5.2.5	Representation Dimension.	116
5.3	Schema Definition in OWL	118
5.4	Inter-schema Mappings Definition in OWL	126
5.5	Chapter Summary	133
6	System Modeling Issues	135
6.1	UML modeling	135
6.1.1	Use Case introduction and definitions	135
6.1.2	Use Case: ICATool	137
6.1.3	Robustness diagram for the ICATool	139
6.1.4	Activity diagram for the ICATool	140
6.2	SEAM modeling	143
6.3	Chapter Summary	148
7	Conclusions and Future Issues	149
7.1	Contributions of the Thesis	149
7.2	Future Directions	150

List of Figures

1.1	State of affairs.	2
1.2	Integrated solution.	3
2.1	Generic Structure of an Interoperable System.	10
2.2	Mediator-based approach.	10
2.3	Agent-based approach.	12
2.4	Kinds of ontology from [Gua98].	15
2.5	Spatial Relationships Transitions.	27
2.6	Allen’s interval relationships.	28
3.1	MADS structural notation.	34
3.2	MADS dimensions: structural.	35
3.3	MADS basic hierarchy of spatial abstract data types.	36
3.4	MADS dimensions: structural & spatial.	36
3.5	MADS basic hierarchy of temporal abstract data types.	37
3.6	MADS dimensions: structural & spatial & temporal.	37
3.7	MADS dimensions: topological relationship.	38
3.8	Perception varying object type Round Cross	39
3.9	Two mono-perception object types linked with the correspond inter- representation link.	40
3.10	Schema S_1 : Park Administration.	41
3.11	Schema S_2 : Road Administration.	42
3.12	Schema T_1 : A City for Tourists.	42
3.13	Schema T_2 : A City for Tourists.	43
3.14	Integration Methodology.	45
3.15	Modeling Concept Sets for the same object.	47
3.16	Population relationships.	47
3.17	Sample integration patterns.	48
3.18	Cadastral plans.	50
3.19	Structural patterns for the disjoint operator between the populations of the related modeling concept sets.	50
3.20	A fragment of Lausanne city map.	51
3.21	Structural patterns for the intersection operator between the popula- tions of related modeling concept sets.	52
3.22	Structural patterns for the inclusion operator between the populations of related modeling concept sets.	54
3.23	Structural patterns for the equality operator between the populations of related modeling concept sets.	55
3.24	Another structural pattern: partition multiple specification.	55
3.25	IC compatibility verification.	60
3.26	Validation block diagram for two object types A and B with a common attribute c	61

3.27	Integrated solutions obtained with different patterns.	64
3.28	A fragment of a domain ontology.	66
4.1	Integration phases.	70
4.2	Topological relationships.	73
4.3	Population relationships and corresponding DL expressions.	75
4.4	Schematic solutions under the intersection relation between the pop- ulations of the source schemas for integrated schema T_{int}	80
4.5	Multi-representation solutions under the intersection relation between the populations of the source schemas for integrated schema T_{int}	82
4.6	Multi-representation solution for bus lines and bus stops in an T_{int}	83
5.1	A non-conforming to the SADT hierarchy schema element.	86
5.2	Integrity constraint for the ternary relationship along from Figure 3.10	99
5.3	Decomposed ternary relationship along from Figure 3.10	99
5.4	Transformation for relationships with attributes.	105
5.5	OWL: Geo types for basic spatial instances.	106
5.6	OWL: spatial data types hierarchy.	108
5.7	OWL: temporal data types hierarchy.	110
5.8	OWL: Interval temporal type.	111
5.9	MADS: topological relationships, validity table.	112
5.10	OWL: adjacent topological property.	113
5.11	MADS: synchronization relationships, validity table.	114
5.12	Disjoint synchronization relationship	115
5.13	Temporal equal property.	115
5.14	Perception stamps T_1 and T_2 with the corresponding classes	117
5.15	Perception stamps t_1 and t_2 with the corresponding classes	118
5.16	Some spatial classes from schema T_1 from Figure 3.13 defined in Protégé OWL.	119
5.17	OpenTime attribute of the Museum object type from Figure 3.13 mod- eled in Protégé.	120
5.18	Schema T_1 in Protégé.	122
5.19	Schema T_2 in Protégé.	123
5.20	Validation for the <code><owl:equivalentClass></code> condition.	128
5.21	Constraint for the topological properties.	129
6.1	The Use Case for the ICA Tool.	138
6.2	Robustness Diagram Elements.	139
6.3	Robustness Diagram Rules.	140
6.4	Robustness Diagram for the ICATool.	141
6.5	Activity Diagram for the ICATool.	142
6.6	Business Level Model	144
6.7	Operation Level Model	144
6.8	BPMN elements.	145

6.9	BP diagram for the ICATool.	147
-----	-------------------------------------	-----

List of Tables

2.1	How a 'building' object can be represented.	8
2.2	Spatial and Interval Relationships	29
3.1	Integration Phases	32
3.2	MADS topological relationships.	38
3.3	MADS synchronization relationships.	39
4.1	Conceptual model element description	81
5.1	Structural Dimension - MADS vs OWL	105
5.2	Spatial Dimension - MADS vs OWL	109
5.3	Temporal Dimension - MADS vs OWL	111
5.4	Constrained relationships - MADS vs OWL	116

Chapter 1

Introduction

1.1 General Context of the Study

This thesis proposes to address the well-know database integration problem with a new method that combines functionality from database conceptual modeling techniques with functionality from logic-based reasoners. Let us first illustrate the benefits of database integration by describing a typical imaginary situation where data is shared among different organizations. Let us assume an institution InstA requires acquiring additional information to make its database DataA conform with the latest administrative instructions. The traditional scenario to implement such an update is depicted in Figure 1.1 and is described as follows:

- the additional data required by the institution is already digitized in database DataB, owned and used by another organization InstB;
- the data formats used by InstB and InstA are different because the information systems used by these institutions were designed separately; DataA and DataB are quite probably incompatible;
- DataA and DataB might be overlapping, but there is no global knowledge on what information is available in both data sets;
- under the condition that the owner of the needed data is known, these data can be exchanged by means of communication devices such as telephone, fax or e-mail.

In such a situation InstA might decide to conduct its own data acquisition independently from InstB, or request available data from InstB. In both cases, the acquired data should be (re)modeled and merged with the DataA data set. A data acquisition procedure organized in such a way repeatedly requires considerable time and human resources.

On the other hand, these two institutions might have chosen an *integrated* data management solution, which is shown in Figure 1.2. This solution is characterized

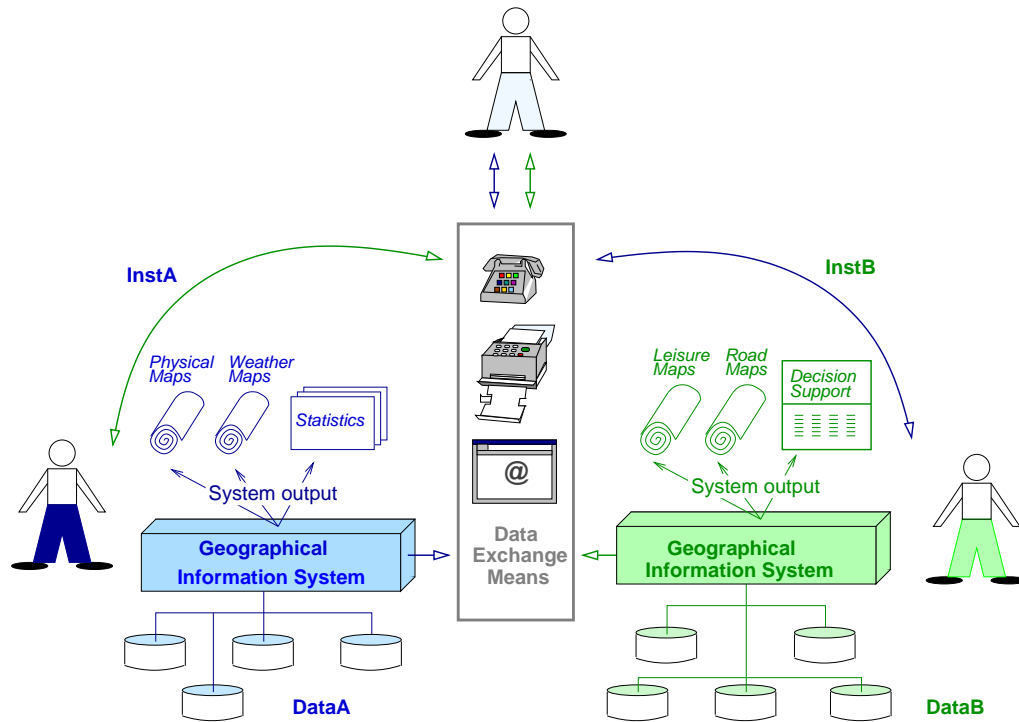


Figure 1.1: State of affairs.

by the existence of a common view over the data in DataA and DataB. Under this assumption, the participating institutions can obtain the following advantages:

- the integrated data set, let us call it DataAB, is available to both institutions via an adopted common data model. This makes data sharing much easier. InstB can benefit from the information collected by InstA and vice versa;
- the process of building DataAB includes a process of data cleaning by comparison, aimed at removing errors and improving the quality of the data sets. The process of data cleaning can be seen as a preliminary step or as a by-product in the move towards creating an ontology for the involved institutions;
- the integration process also includes a conceptual schema design phase. A conceptual schema conveys the user understanding of the universe of discourse of the application. This conceptual representation is an important achievement in itself as it holds the semantics of the data and is invariant to system implementation. The common conceptual schema (usually denoted as the federated schema) offers a global and complete view of the participating data sets;
- the initial systems and data sets are preserved for local usage; thus, all the existing applications remain operational;
- the functionality of the new federated Geographical Information System (GIS)

fully includes the functionalities of the former GISs and may be augmented by additional features;

- allowing Web access to the cleaned and semantically clear DataAB set provides for more features. For example, an on-line implementation may include a Map-on-demand feature, in which a user may define a required map type, based on a variety of information available through the DataAB set, to be generated for him/her on-line.

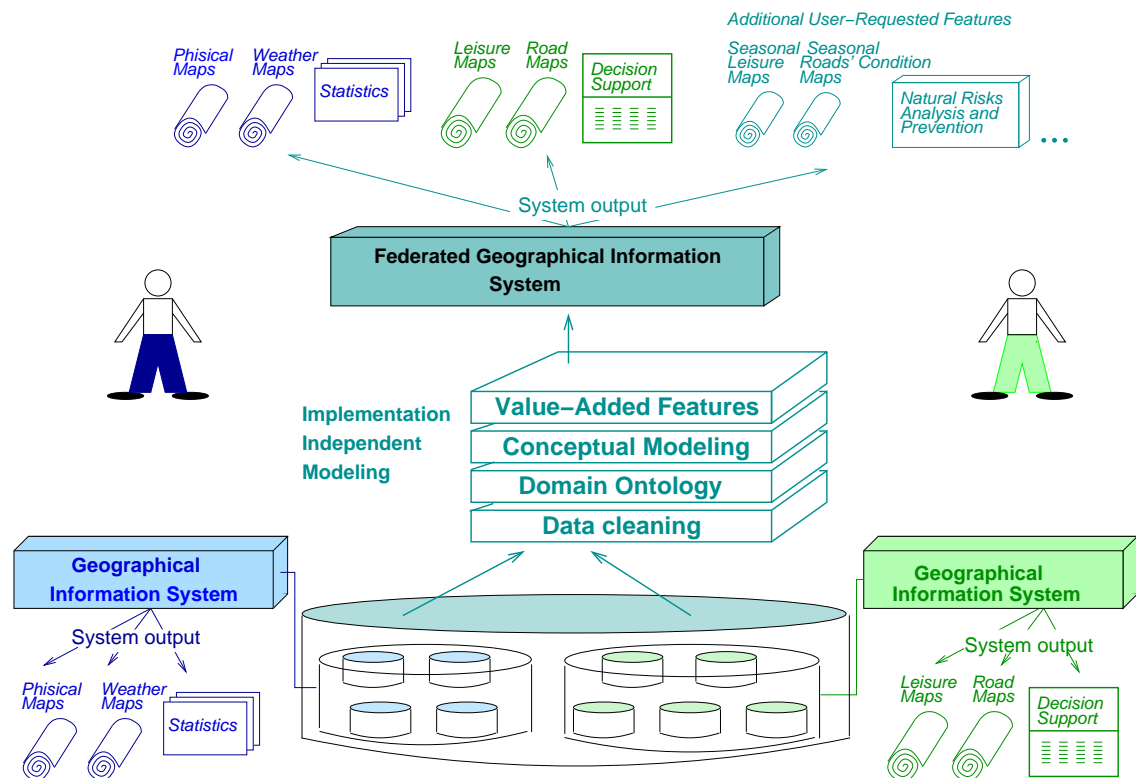


Figure 1.2: Integrated solution.

Looking at the human dimension, the construction of an integrated data set and a federated GIS on top of existing systems requires not only designer skills but also active participation and willingness by the users of the local systems. Nevertheless, the investment in building integrated data sets is justified by an improvement in the collaboration between institutions, as they establish a common data set and vocabulary during the federated conceptual schema design process.

Our vision of a viable way for transparent and meaningful processing of heterogeneous spatio-temporal data is to put data semantics at the foundation of the integration process. In this thesis we present and correlate various means of integration, showing how they can be organized as components of the mediation level of an inter-operable system. Because our target domain is the integration

of spatio-temporal databases, the integration method we propose relies on the use of a powerful spatio-temporal conceptual data model. Using as example several spatio-temporal schemas, the thesis develops an integration methodology based on semantic inter-schema mappings and multiple integration policies.

1.2 Motivations

At the dawn of the Information System (IS) design era, when the volumes of information managed by applications were calculated in megabytes, the modeling of the semantics (from the Greek *semantikos*, or 'significant meaning') of the data was not a crucial issue. Very often, the conceptual modeling steps were just omitted. This attitude to the system modeling process was determined by specificities of small-scale systems design. Due to simple data structures, the content of a data set or a (relational) database could be understood, for example, from the names of the tables or the names of the fields. The effort needed for conceptual modeling, and the time spent in it, were considered inefficient, if not useless, compared to the expected little time needed to intuitively understand the content of a data set by just examining the names in the database structure. Another property of systems of that time, nowadays called legacy Information Systems, was their mostly local usage and a relatively short life cycle. Information Systems' users were able to actually consult the system designer in case of problems or misunderstandings. In other words, there was a popular feeling that the complexity of the conceptual knowledge could be managed at the human scale and on an intuitive level.

Nowadays, the issues in Information Systems' design and exploitation have shifted. Modern ISs manage terabytes of distributed heterogeneous data, servicing diverse application domains that include nearly all aspects of everyday life. Computerization ranges from book and music private archives to digital descriptions of entire countries including satellite maps and detailed cadastral plans. Data collection process becomes more and more available thus encouraging data owners to share their data and provide services based on the collected data. Clearly, the boom in information flow and exchange mandates that the description of the data content should now be presented in an expressive semantic form, so that it could be understood by other humans or computer agents. Due to the diversity and the multitude of data sources, in many organizations the question of collaborative data usage becomes crucial. Some examples are the reuse of different statistic data collected by different hospitals; a system managing driving license holders data across several countries; or a system that allows for a collaborative administrative data usage concerning the same land (or country). All these examples assume that there is a way to integrate the underlying data sets at the instance and schema levels. Given the complexity of data, caused by the diversification of the services provided by ISs, there is a need for powerful integration methodologies. Knowing only the name of database tables and attribute descriptions is no longer enough to convey the semantics of the pop-

ulation of a database. Integration methodologies should be driven by and focused on the semantics of the data sets, with the semantic description shifted up to the conceptual level.

In general, the semantics of a database is described by a conceptual model, which we therefore regard as a core element in integration methodologies. This argument is even more evident if we consider the domain of geographic applications, where the data describing given geographic phenomena are collected and used for quite different purposes by different data owners. Designing an integration methodology for geo-data requires that an expressive conceptual model be used as the *common* data model. This common model should allow representing a variety of features characterizing geo-data and geo-applications, including those currently managed by Geographical Information Systems (GIS). Such features include, for example, space-varying object properties (e.g., the depth of a lake), evolution of geo-objects in time (e.g., reconstruction of a road network), various relationships enriched with topological and synchronization semantics (e.g., a school should be located inside a residential area, and construction of that school should be finished before the residential area is put into operation).

Such complex relationships and data properties suggest another distinctive property of integration methodologies for complex data. An automated process to discover correspondences between different data sets would require a very sophisticated and resource consuming heuristics to lead to high precision results. This last observation again highlights the need for a very expressive conceptual model to provide for a precise and human-understandable way of expressing data descriptions and correspondences' definitions. Notwithstanding, verification and deduction processes should be used to validate the data description and the correspondences provided by the designer of an integrated system.

1.3 Outline of the Thesis

We start our thesis by overviewing architectures for inter-operable system in **Chapter 2**. We position our methodology as adhering to the federated information systems approach. Accordingly, our integration process results in an integrated schema that is called a *global* schema in a federated system. By choosing the federated system approach, we also conform to one of the important operational particularities of ISs: durability. In a federated system, the component legacy applications and data remain functional, which facilitates the acceptance and transition to the new system (for the users of the legacy systems). In our methodology we thus do not consider the data integration level, the methodology is designed for the conceptual schema level.

We develop in **Chapter 3** a flexible integration approach where the integrated schema designer has several viable ways to construct a final integrated schema. For different related schema elements we provide the designer with four general policies

and with a set of structural solutions or structural patterns within each policy. To always guarantee an integrated solution, we provide for a preservation policy with multi-representation structural pattern.

To state the inter-schema mappings, we elaborate in *Section 3.4.2* a correspondence language whose syntax is based on the syntax of the common data model. With the inter-schema mappings, the designer can state correspondences between related populations, and define the conditions that rule the matching at the instance level. These matching rules can then be used in query rewriting procedures or to match the instances within the data integration process. We associate a set of putative structural patterns to each type of population correspondence, providing a designer with a patterns' selection for integrated schema construction. The correspondence language has three facets: structural, spatial, and temporal, allowing to relate the thematic representation as well as the spatial and temporal features. We provide an algorithm that checks the completeness of the set of inter-schema mappings.

Employing an expressive common data model, we assume that the schema designer is knowledgeable enough to state the inter-schema mappings manually. But we cannot assume that the designer states all the mappings, and that they are all correct. To cope with this issue, we propose in **Chapter 4** a validation approach to ensure validity and completeness of the inter-schema mappings set. Validation calls for using another data modeling approach that, contrary to conceptual models, was developed to provide for an inference mechanism. We couple conceptual models with description logics, employing each approach for the purpose it is best suited for, i.e., conceptual modeling for semantically rich data description, and description logic to use its inference mechanisms for validation of manually stated assertions.

Inter-schema mappings cannot be validated autonomously, i.e., they are validated against the data model and the schemas they link. Thus, to implement our validation approach, we translate the data model, the source schemas and the inter-schema mappings into a description logic formalism, cf. *Section 5*. The added value of the complete translation (i.e., including the data model and the source schemas) is that we can validate not only the inter-schema mappings, but also the compliance of the source schemas to the data model, and infer implicit relationships within them. As the result of the validation procedure, the schema designer obtains the complete and valid set of inter-schema mappings and a set of valid (flexible) schematic patterns to apply to construct an integrated schema that meets application requirements.

In **Chapter 6** we describe system modeling issues. We consider two system modeling approaches, UML and SEAM. We design high level system diagrams, i.e., Use Case and Activity in the UML notation, and Business and Operation Organizational Level models in SEAM notation. We compare the two approaches by the comprehensibility and traceability of system diagrams. **Chapter 7** concludes the thesis and outlines prospective developments for our integration methodology.

Chapter 2

Relevant Research Areas: State of the Art

There are several domains that are relevant to the design of an integration methodology. In this chapter, we first describe the system architectures that were considered for the choice of the final integrated solution (Section 2.1). Then, we continue with the discussion on the two most widely used data modeling approaches: conceptual modeling and modeling with ontologies, where we argue that the two approaches are complementary to each other (Section 2.2). Different inter-schema mapping approaches are presented in Section 2.3; we choose the composing data models and services that best suit what we intend to do in our methodology and then design a hybrid approach that exploits their strong features. One of the innovative features of our methodology is the validation support during the integration process. Therefore, we describe available validation services in Section 2.4. Finally, in Section 2.4.3 we discuss the formalisms we use to represent spatial and temporal data, based on their suitability for our validation purposes.

2.1 Interoperable System Architectures

Interoperability arises as a problem in heterogeneous systems where different data resources coexist and there is a need for meaningful information sharing in the system. The heterogeneity of the data can be due to semantic, syntactic, and structural differences of the data sources. One of the demonstrative realms of diversity of data representation is the spatio-temporal domain. In the spatio-temporal domain the same objects can be represented (and are represented) from multiple and greatly diverse points of view. For example, a building can be represented from four different points of view as shown in Table 2.1.

Because of the larger diversity of its users and usages, spatio-temporal data heterogeneity is usually higher than thematic data heterogeneity. Traditional databases serve a restricted community of users, still most frequently within a single organization. Their users tend to use a common vocabulary and tend to use some common

Table 2.1: How a 'building' object can be represented.

<i>Purpose of representation</i>	<i>User</i>
Architectural style and its fitting in the neighborhood environment	Urban planning administration
Robustness of the construction of the building and the materials it is built of	Rescue crew of the city
Condition of the building and suitability for living in it	Renovation construction company
Location and dimensions of the building	Cadastral department of the city administration

information. This makes semantic matching easier than for geographical databases, where users of the same database share very little vocabulary and few information from one application to the other. Thus, establishing correspondences between attribute value domains, for example, is definitely insufficient to solve the semantic matching issue for geographical data sets. An adequate amount of integration work has to be done before we can establish correspondences on the attribute domain level. As illustrated for example in Table 2.1, to propose rules by which it can be inferred that the two or more different data representations portray the same object from the real world is a challenge. Such rules, or correspondence assertions, are an appropriate means to identify common populations, and common spatial, and/or temporal features of objects from different applications. Derivation of semantically driven correspondence assertions is feasible if the data model used for the application domain is sufficiently semantically expressive to convey the knowledge on which the derivation process relies. This implies that the application data should be remodeled or pre-integrated at the conceptual level¹ in a semantically rich model. Such a model is called *Common Data Model* (CDM). A CDM should have a minimal number of concepts while being sufficient to capture the semantics of the application domain and tasks to which it is dedicated.

We introduce the research domain of interoperability by presenting a generic view on interoperable systems and proceeding by refining the scope of possible architectures to agent-based and mediator-based systems as these two architectures are most widely adopted by the research community. In Section 2.1.1 we discuss the main features that can be accomplished within each architecture and point to the one that is more suitable for our domain of interest. Within this architecture in Section 2.2.1 we define the system components for which we contribute some proposals of our own.

¹as the implementation independent level

2.1.1 Interoperable System Components

Generally, an interoperable system consists of three main components as shown in Figure 2.1. At the **foundation** level there are heterogeneous legacy data sources. The **mediation** level supports exchange of queries and results between legacy data sources and applications. At the **application** level the interaction with the users is carried out [GMZB99].

Without the 'Value-Added Services' layer, the structure presented would be an ordinary information system architecture designed for a particular group of users operating a specific set of data sources. Nowadays, modern information systems increasingly address information and knowledge acquisition issues over heterogeneous data sources [GEFK99, She99], and traditional simple architectures are no longer an answer for an information system architecture. An information system with an intermediate level between 'USERS' and 'SOURCES' levels is called mediated. The mediation level provides the users with services based on the data previously collected and operated for other purposes, and within other information systems, entitling such systems to be defined as interoperable. In the literature many different implementations of the mediation level can be found [QL94, AOTT98, BBB⁺98, CR99, AM99]. The components emerging from these implementations are the following:

- *application ontology* - a structure containing concepts and their hierarchy for the application domain;
- *agents* - intelligent components that can serve different purposes in the system, for instance, locating appropriate data sources in a distributed system, matching user requests with the services available;
- *translators* - components that translate user queries into queries to a CDM;
- *wrappers* - components that translate the heterogeneous source data into data manageable via a CDM;
- *integrators* - components that perform integration of heterogeneous data sources based for example on an ontology, or a CDM;
- *mediator* - a complex component that provides the application level with transparent access and processing over a set of heterogeneous source data.

The mediation level can incorporate a set of different components. The choice of these components and functionality at the mediation level is driven by the intended objective of the system. In the sequel we will present two system architectures that support very different functionality: mediator-based [AOTT98] and agent-based [FPNB99] systems.

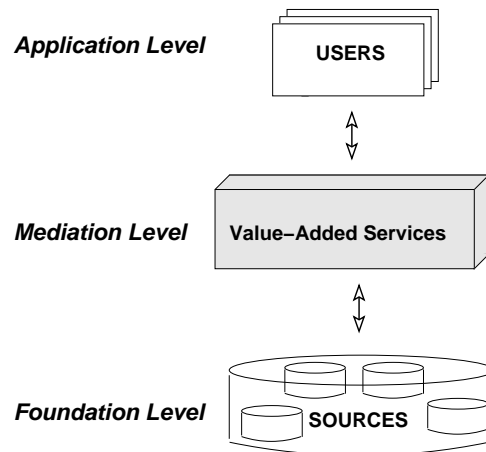


Figure 2.1: Generic Structure of an Interoperable System.

2.1.2 Mediator-based systems

In a mediator-based system it is assumed that there is a component to which all the user's queries are addressed, where these queries are processed, and where the results of these queries are sent back to users. This component plays the mediation role between the users and data sources and maintains the global vision of the system [Wie92]. The mediator-based system that we have chosen as illustrative example is presented in [AOTT98].

Figure 2.2 shows a simplified architecture of the system. As the basis for data integration, a CDM was used. The authors have chosen the object-oriented data model whose capabilities in modeling semantics and relationships were suitable for the application area.

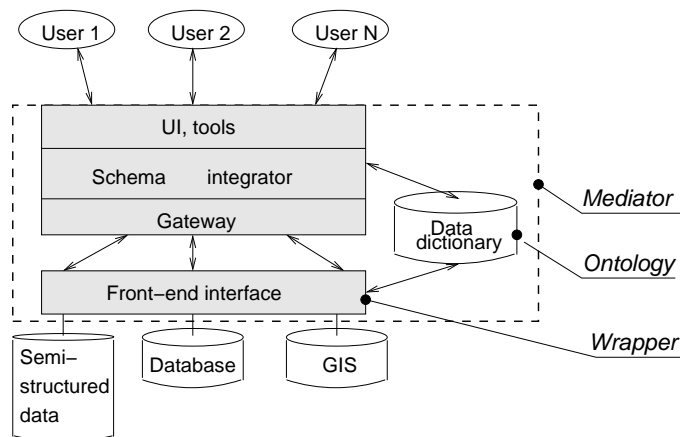


Figure 2.2: Mediator-based approach.

The local schemas of component databases are translated into the CDM and are enriched semantically if necessary. The federated schema is a schema constructed in

CDM based on the user specifications on the subset of data of interest to them. Thus, the users view the system as a single database containing the data they requested. User queries are directed to the mediator component of the system where the queries are decomposed and then translated to the local schema query languages. Although the technique described in the paper suits the application requirements, the authors do not address issues such as semantic conflicts resolution and integrity constraints management. In addition, a disadvantage of the system is that the component databases are not operable locally and that the data sources updates are done as well globally. Presenting the system capabilities the authors mention that:

... the schema integrator provides facilities for integrating the schema exported from the component databases into the federated schema. It needs to generate mapping between the exported and federated schemas and must have a reasonable capability for detecting conflicts between data...

However, no real example of the schema integrator functionality is given in the paper. The authors propose the use of a mapping table² for matching object representations.

2.1.3 Agent-based systems

As an example of an agent-based architecture we consider the InfoSleuth system presented in [FPNB99]. InfoSleuth is a distributed system where the data sources and the users reside on different sites and are connected by sets of different agents. System agents communicate on the basis of a system ontology, which is the only global component of an agent based system. *Ontology* is a specification of how to represent the objects, concepts and other entities that are assumed to exist in some domain of interest and the relationships that hold among them [dic]. The InfoSleuth ontology does not represent a global structural vision of the system data sources but only the set of terms the system is aware of.

In an agent system three main agent types can be pointed out [FPNB99]:

- *User* agent - maintains the user state and provides the system interface that enables the user to communicate with the system independently of the user location.
- *Resource* agents - translate queries and data stored in some external data repository between their local forms and their representation (data model) in the system.
- *Broker* agents - match requests for services from user agents with resource agents that can provide them.

²which can be seen as a simplified ontology

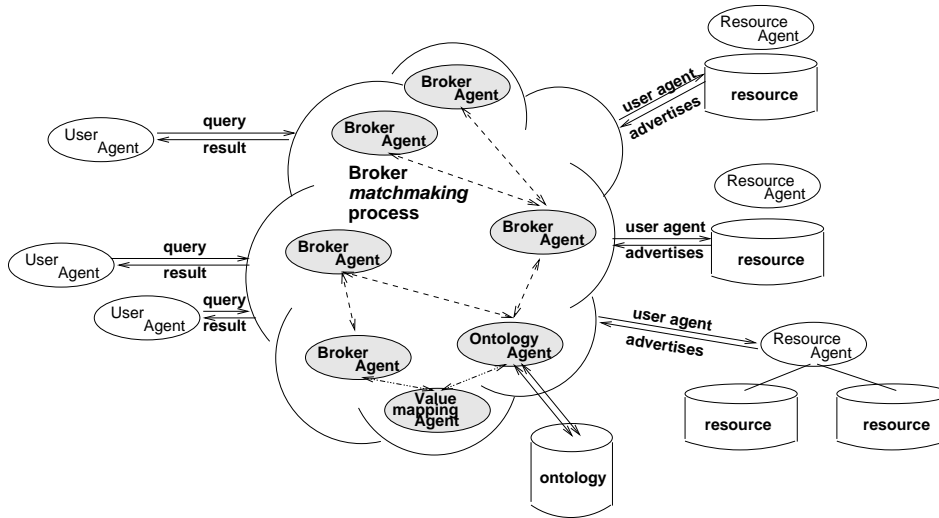


Figure 2.3: Agent-based approach.

- *Resource, Ontology, and Value Mapping* agents - are the means of interoperability of the system.

All the types of heterogeneity, i.e., structural, syntactic and semantic, are solved by resource agents and value mapping agents in InfoSleuth based on the developed ontology. The value mapping agents map query terms to and from the common value domain which is defined by the system ontology. The common value domain reduces heterogeneity only in terms of allowed attribute values but not in terms of data representation. Users query and view data in whichever value domain they prefer, and their user agents perform the value mapping necessary to communicate with other agents in the common value domain. To perform a value mapping a user agent contacts a value mapping agent. Thus, all the operations related to data interoperability are done through consultation with the ontology of the system. Referring to Figure 2.3, it can be seen, that the functionality of the *user* and *value mapping* agents is similar to that of the *query translator*. On the other hand, a resemblance can be found in *wrapper's* in Figure 2.2 and the *resource agent* functionalities.

The partial knowledge of the available data and its location is maintained by broker agents. The information stored by broker agents is partitioned in a way that the whole set of broker agents 'knows' about all the data available in the system³. The system ontology stores the hierarchy of the data the system is aware of. When the user queries the system, it is the broker agent(s) functionality that determines whether the data requested can be found in the system data sources. The user does not have a global view of the system data and does not know whether his/her request can be met. The partition of the knowledge and communication between the broker

³Depending on the system design redundancy may be allowed or even required.

agents ensures that the user agents and the resource agents are fully connected, e.g., any user can potentially reach any resource.

As it follows from the system description the users are assumed to pose **SELECT**-type queries. This system does not allow **UPDATE**-type queries. The broker agents are oriented towards locating requested data, matching the semantic and syntactic information of the user agent and a data source. Done automatically, the matchmaking process restricts the amount of semantic information that agents can operate. The last observation together with the absence of a global vision of the system limits the application area of agent-based systems.

2.1.4 Summary

Comparing the approaches presented above, we bear in mind the following characteristics of an intended interoperable system:

- at the foundation level there are heterogeneous spatio-temporal data sources;
- at the application level there are users with their vision of the universe of discourse;
- users expect transparent operations on geo-data stored in different formats, with different resolutions, and for different purposes.

Let us first briefly summarize the pivotal characteristics of the two approaches to make our reasoning about their applicability to spatio-temporal domain more clear.

- *Agent-based*
 - the system ontology is the only global component in the system,
 - the users do not have a global view of the system,
 - updates are allowed on the local level and may not reflect the system ontology.
- *Mediator-based*
 - a mediator stores the schemas of component databases and the relationships between them or a federated schema of the system depending on the implementation,
 - users have a schematic partial or global view of the system,
 - updates are theoretically allowed from both the global and local levels, but with no clear methodology for update propagation, the global consistency of the system is an open question.
- *Desired interoperable system characteristics*

- a global schema is constructed based on semantic and syntactic information, conflicts are resolved at the global level;
- consistency of the data is ensured during the integration process;
- updates are allowed from the global level as well as from the local level, consistency of the component databases is ensured by an update propagation mechanism.

The agent-based system is hardly a viable way to accomplish such a task. Agents are more oriented towards determining the location of data sources in a distributed system: the data sources available are heterogeneous in the sense that different objects are stored in different locations. Whereas, dealing with the spatio-temporal data it is also likely that the same phenomena would be presented in different ways. Consequently, to establish a relation between different objects, a semantically based methodology should be employed which requires semantically rich integration platform. Semantic information carried by agents in the InfoSleuth system is not sufficient for integration of spatio-temporal objects.

Moreover, in the spatio-temporal domain the global vision of the data and data structure are indispensable properties of the system. An attempt to augment the broker agents with more semantic information would lead to an increase in the system response time and therefore to a decline in the system performance. Another feature of the agent-based system architecture is that the matching process is automatic, which is potentially incompatible with spatio-temporal integration process. This advantageous feature of the agent-based architecture is not yet applicable to the spatio-temporal domain. To the best of our knowledge, there is no methodology which would support an automatic matching process for spatio-temporal objects modeled within different applications.

More features that can be adopted in the spatio-temporal domain are borne by the mediator-based approach. The example given in Table 2.1 suggests that for integration purposes the data sources should be implemented or translated (on the conceptual or physical level) into a model which allows to express diverse semantic aspects of the data objects. In addition there should be an expressive language which allows to define interrelations of spatio-temporal object types. In Section 3.4 we present our integration methodology in more detail.

One of the common system components in Figure 2.2 and Figure 2.3 is the system ontology. The notion of ontology is not unambiguously perceived by the database community, whereas ontology plays a key role and for particular implementations is the only mean of integration of the system data sources. In the following section we present a notion of ontology and conceptual models as the next level of abstraction of an application area.

2.2 Ontologies and Conceptual Models

Guarino [Gua98] distinguishes several levels of ontology, as shown in Figure 2.4. The top-level ontology is a representation of the 'truth', i.e., the representation of the real world however, no specific use is implied in it. The top-level ontology is the most generic type of ontology where concepts like space, time, matter, object are presented. The taxonomy of top-level ontologies is the simplest one from the structural point of view, as the only relation that is used for top-level ontologies is subsumption. Thus, a top-level ontology has a non-cyclic tree structure without multiple inheritance. An example of such ontology can be found in [GW00b].

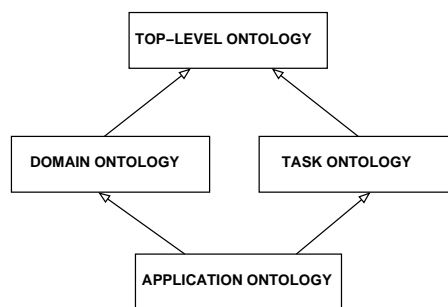


Figure 2.4: Kinds of ontology from [Gua98].

The reasoning behind constructing a top-level ontology lies in the four meta-properties borne by the things of the real world. These meta-properties are: *rigidity*, *identity*, *unity*, and *dependence*, as described in detail in [Gua98] and [GW00a]. In brief, identity is related to the problem of distinguishing a specific instance of a certain class from other instances by means of a characteristic property, which is unique for it. A rigid property is a property that inherently always holds for all the instances of a class. For example, assume two classes PERSON and STUDENT. From the point of view of rigidity, PERSON is rigid - all the instances of this class are of PERSON type, on the other hand, STUDENT is not rigid - the same individual can be STUDENT in one context, and non-STUDENT in another. From the previous example we can conclude that rigid classes supply the identity, and non-rigid ones just carry an identity. Unity is related to the problem of distinguishing the parts of an instance from the rest of the world by means of a unifying relation that binds them together. An example identity query would be 'What is this country?' and a unity query would be 'Does this canton belong to this country?'. If existence of an instance of a class implies a necessary existence of an instance in another class, then the former possesses the dependence meta-property. An example would be a CANTON class that implies existence of a COUNTRY class. Ascription of those properties to classes imposes certain constraints on the positional relationship of these classes in the top-level ontology. For example, one of the imposed structural constraints is that a dependent class cannot subsume the class it depends on. The role of the top-level ontology is to formalize the real world in a widely sharable,

multidisciplinary way to be used further as the pattern for different domain and task ontologies. For the spatio-temporal domain, the top-level ontology is one of the subjects of research and agreement of the Open Geodata Consortium [OGI].

When an ontology contains some domain specific concepts or concepts related to general features of an application we step down the ontology hierarchy. Such an ontology is a subjective, refined representation of the same concepts as in the top-level ontology. Domain and task ontologies already can be used in the integration process.

2.2.1 Ontologies vs Conceptual Schemas

Domain and task ontologies contain the classes that are further used in the conceptual schemas. When we start to model the roles of the domain ontology classes and methods associated to them, we are at the level of application ontology, the most specific and application dependent type of ontology. The main thread through-passing the structure shown in Figure 2.4, is that domain, task, and application ontologies are structurally compliant with the top-level ontology.

The objective of conceptual modeling is to represent application data together with the rules of the application domain. In other words, conceptual models allow to represent the user understanding of the universe of discourse. The task of designing a modern information system becomes more complex with the progress of information technology and with the users becoming more demanding for the functionality of the information systems. In such circumstances conceptual modeling gains in importance, as it is the starting point for understanding of the user needs. The most important properties that a conceptual model should have are: abstraction, non-ambiguity, ease of understanding and verification, and implementation independence [JM00]. The last property implies that a conceptual model should be expressive enough so that the same conceptual schema would be valid even when software paradigms were upgraded or replaced. Conceptual schemas, being the representation of the user perception of the application domain, constitute a good basis for integration of heterogeneous domain sources in an interoperable system.

An ontology is the representation of real world without bearing in mind any application of this representation; conceptual schemas are implementation independent representations of the application area, containing users vision of the application domain. The link between an ontology and a conceptual schema is that the conceptual schema of an application domain should be structurally compliant with the ontology for the same domain. Nevertheless, in our research we base our approach only on domain ontologies and conceptual schemas without making any reference to a top-level ontology as we are not aware of the existence of an approved top-level spatio-temporal ontology. In the following section we discuss the choice of the spatio-temporal common data model.

In our integration methodology, presented in Section 3.4, we chose a conceptual representation as the starting point for matching heterogeneous data sources. The

choice of an appropriate conceptual model depends on the completeness of representation allowed by it, its formal semantics and its simplicity of use and interpretation.

2.2.2 The MADS Conceptual Model as a Common Data Model

Applications manipulating geo-data are difficult to model due to the particularity and complexity of the spatial and temporal components. More facets of real-world entities have to be considered, e.g., location, form, size, time validity; more links are relevant, e.g., spatial, temporal links; several spatial abstraction levels often need to be represented. Thus, modeling spatio-temporal databases requires advanced facilities [SPV00], such as the following.

- Objects with complex structure (e.g., composition/aggregation links, generalization links), at least equivalent to those supported by current object-oriented models. This should achieve full representational power in terms of data structures;
- Alternative geometry features to support both discrete and continuous views of space, representations at different scale/precision, multiple viewpoints from different users;
- Spatial objects with one or several geometries associated to different resolutions or user points of views;
- Temporal objects with complex life cycles that allow users to create, suspend, reactivate, and eventually delete objects;
- Timestamped attributes that record their past, present, and future values;
- Spatio-temporal concepts for describing moving and deforming objects;
- Explicit relationships to describe structural links as well as spatial links (such as adjacency, inclusion, spatial aggregation) and synchronization links (such as before, during). The knowledge of the topological links between real-world entities is an essential requirement for applications.
- Causal relationships describing the causes and effects of changes that happen in the real world.

The model must also allow defining schemas that are readable and easy to understand. A key element for achieving this double objective is the orthogonality of the structural, temporal, and spatial dimensions of the model (and more generally of the concepts of the model). Thus, whatever the concept of the model, e.g., object, relationship, attribute, aggregation, the spatial and temporal dimensions may be associated to it.

In our research we use the MADS conceptual data model [PSZ99, PSZ06] as the *common* data model (CDM). MADS stands for Modeling of Application Data with Spatio-temporal features. In [PSZ99], the authors analyze different spatio-temporal data models along the axes of expressiveness, simplicity and comprehensiveness, formalism, and user friendliness, making the conclusion that none of the existing models satisfied all the requested criteria. MADS includes a set of predefined spatial and temporal Data Types (DTs) that are used for describing the spatial and temporal extents of the spatio-temporal elements of schemas. Spatiality and temporality may be associated to object and relationship types, aggregation links, and attributes. The fact that MADS structural, spatial, and temporal domains are orthogonal entails that spatial and temporal features can be freely added to any schema designed in MADS.

2.3 Inter-Schema Mappings

Information sharing between heterogeneous information sources is a great challenge which has been the focus of various works but still remains an open problem. The earliest investigations addressed database schemas integration, i.e., merging a set of given schemas into a single global schema ([BLN86, EP90, SL90, PS98]). This problem has been studied since the early 1980s. It arises in building a database system that comprises several distinct databases and in designing the schema of a database from the local schemas supplied by several user groups. The integration process requires establishing semantic correspondences between the component schemas and then using the matches to merge schema elements.

Enabling the cooperation of heterogeneous information systems is not easy to achieve because related knowledge is most likely described in different terms and using different assumptions and different data structures. Heterogeneity may arise from syntactic, structural and semantic differences in the data sources. *Syntactic heterogeneity* may come from the use of diverse database models (e.g. object oriented vs. relational), *structural heterogeneity* arises from different conceptual choices during the database modeling phase (e.g., modeling something as an object, as a relationship, or as an attribute), and *semantic heterogeneity* deals with differences between the terms used to represent information and their intended meaning. Heterogeneity is even higher for spatio-temporal data, due to the existence of two very different paradigms for data representation, known as the raster mode (space is represented through images) and the vector mode (space is represented as sets of localized objects). Moreover, spatio-temporal data may be represented using different granularities or levels of detail for the spatial and/or temporal features. Also, for geographical data, we have to consider topological relationships between objects, temporal evolution and synchronization relationships.

Irrespectively of the system architecture, being able to recognize corresponding information in heterogeneous data sets and to describe the mappings in between is

a fundamental task to enable the cooperation. A large number of papers have investigated various facets of mappings, such as mapping discovery, mapping definition or mappings usage.

2.3.1 Mapping discovery.

Works on mapping discovery aim at providing heuristics to find corresponding elements in different information systems. Such works basically rely on similarity measures. This topic is particularly important when considering cooperation between ontologies where automatic discovery of mappings is crucial due to the huge number of concepts. M. Ehrig and Y. Sure [ES04] propose a methodology combining different similarity measures for identifying mappings between two ontologies. A. Doan et al [DMDH04, DMDH02] propose a system, GLUE, that apply machine learning techniques to improve the mapping discovering process. However, complex mappings have proven difficult to extract and the mapping discovery procedure certainly requires human feedback. R. Dhamankar [DLD⁺04] presents a promising system, iMAP, for the discovery of complex mappings between database schemas. However, mapping discovery between heterogeneous schemas describing spatial data still remains an open issue.

2.3.2 Inter-schema correspondences.

Complementary to those approaches, other research works [CL93, CdGL⁺98, CdGL01, DPS98] do not consider the mapping discovery phase and focus on formalisms to specify and use inter-schema knowledge. From a conceptual perspective, inter-schema knowledge identifies elements (or sets of elements) in two schemas that to some extent, describe the same (or related) facts in the real world, and specifies to what extent the data instances and their type definitions relate to each other (i.e., what is identical, what is similar, what is different). This inter-schema knowledge can then be used to build the integrated schema and to provide for an integrated access to the data sources. The four works presented below follow this objective using different languages. A formalism relying on a logic-based language is proposed by T. Catarci and M. Lenzerini in [CL93]. The language they propose is used to describe both schemas and inter-schema knowledge. The reasoning mechanism of the language can then be used to check inter-schema consistency (i.e., the correctness of the cooperative information system) and to support integrated access to data. D. Calvanese et al. [CdGL⁺98] present an architecture for information integration. A Description Logic called \mathcal{DLR} which includes concepts and n -ary relationships is used to describe the database schemas, to specify the inter-schema knowledge and reasoning services that are used during the integration process. The same language, \mathcal{DLR} , is proposed by D. Calvanese et al. in [CdGL01] to define mappings in a general framework for ontology integration. These mappings allow the mapping of a concept in one ontology to a view, i.e., a query in another ontology. Finally, T.

Devogele et al. [DPS98] propose a complete methodology for spatial database integration based on three phases: schemas preparation, correspondences investigation, and integration. The authors also provide an algebraic data manipulation language (algebra for complex objects) to describe inter-schema correspondences that fully supports the description of correspondences between the spatial features of data.

2.3.3 Querying.

Once mappings are formally defined, one should be able to use them for query answering and reasoning [MBDH02]. D. Calvanese et al. [CdGL01] discuss various approaches for specifying mappings (global- and local-centric approaches) and, for each approach, analyze the complexity of query answering. The authors conclude that mappings should be defined using suitable mechanisms based on query languages. In [HIMT03], A. Halevy et al. express mappings between data sources on a pairwise basis and define inclusion and equivalence relationships between views of each schema. An algorithm enabling queries to go through mappings in order to find data is also proposed. The article "Data Integration: A Logic-Based Perspective" by Diego Calvanese and Giuseppe De Giacomo [CG05] discusses semantic-integration issues in this context, using description logics. It describes an integration architecture where there is a global ontology that contains the information common to the ontologies that need to be integrated. The authors use the power of description logics to answer queries posed in terms of the global ontology with data from local ontologies. They propose a new subset of description logics called DL-Lite, which retains a useful subset of primitives but yields good complexity characteristics.

2.3.4 Semantic enrichment.

In order to reconcile semantic heterogeneity, more semantic information about data is needed. Various proposed approaches add extra information to data either through the specification of meta-data, or through the explanation of the context of data or, more generally, by using descriptions stored in ontologies. *Meta-data* describe the content of the underlying data in an easily understandable way. *Contexts* are more complex descriptions specifying the domain of the source data. *Ontologies*, by definition, provide an encoded representation of a shared understanding of terms and concepts in a given domain and community. They serve as semantic references for users or applications that accept to align their interpretations of the semantics of their data to the interpretation stored in the ontology. Ontologies are actually extensively proposed as a means to overcome interoperability problems [WVV⁺01]. This is the focus of the work of F. Fonseca et al. in [FDC03]. In their framework, conceptual schemas of geographical databases are mapped to spatial ontologies that are considered as the formal representation of the spatial semantics. The objective in describing such mappings is to enrich the conceptual schema descriptions and thus, to improve the integration of database conceptual schemas.

F. Hakimpour and A. Geppert in [HG02] propose a database integration approach that employs formal ontologies merging. Source ontologies (one per database source) are merged by a reasoning system that finds semantic similarity relations between the various definitions used for each concept. An Ontology-based Schema Integrator builds the global schema of the integrated database using the source schemas and the mappings found during the ontology merging process.

F. Fonseca et al. in [FEAC02] propose an Ontology-Driven GIS system which plays the role of a system integrator. The idea is to provide access to data by browsing through ontologies. The architecture is based on four main components namely the *ontology server*, the *ontologies*, *mediators* and *applications* that give access to the information sources. The ontology server is the central component providing the connection between the ontologies, the applications and the information sources. The integration is partly realized by the mediators: when the information system is queried, the mediators extract parts of information necessary to generate a complete instance from the ontologies and the information sources.

Our methodology focuses on the cooperation of spatio-temporal databases. In this respect, our objective is to propose a complete methodology for the integration of spatio-temporal conceptual schemas. Our approach relies on two well-known formalisms: conceptual models and description logics. Spatio-temporal conceptual schemas to be integrated are specified using the MADS conceptual data model [Mur02] which has rich spatio-temporal semantics. Reasoning services of description logics are then used to check the consistency of the mappings that guide the construction of the integrated schema. Compared to the papers presented above, our proposal falls within the scope of approaches that aim at defining a formalism or methodology to specify and use inter-schema knowledge. We do not tackle the issue of mapping discovery as we assume that a set of inter-schema correspondences given by the designer is completed by an inference engine, nor do we consider the subject of query rewriting which is out of the scope of this thesis.

2.4 Validation Approach in Integration Procedures

Once we know correspondences between two sources, we want represent them in a machine-processable way and validate them. Researchers have developed a number of ways to represent mappings declaratively. Some examples include representing mappings as instances in an ontology of mappings, defining bridging axioms in first-order logic to represent transformations, and using views to describe mappings from a global ontology to local ontologies. As we stated earlier, we adhere to a hybrid approach where we exploit two data modeling formalisms: conceptual modeling and modeling with description logic, the latter being used for its formal reasoning support.

2.4.1 Reasoning in Description Logic

Description logics (DL) are a family of knowledge representation languages which can be used to represent the terminological knowledge of an application domain in a structured and formally well understood way. The name description logic refers, on the one hand, to concept descriptions used to describe a domain and, on the other hand, to the logic-based semantics which can be given by a translation into first-order predicate logic. Description logic was designed as an extension to frames and semantic networks, which were not equipped with a formal logic-based semantics. Description logic was given its current name in the 1980s. Previous to this it was called (chronologically): terminological systems, and concept languages. Today description logic has become a cornerstone of the Semantic Web for its use in the design of ontologies. The first DL-based system was KL-ONE, by Brachman and Schmolze [BS85]. Some other DL systems came later, such as BACK [Pel91], KRIS [BH91], CLASSIC [BBMHR91], LOOM [Mac94], FaCT [Hor98] and lately RACER [RAC] and KAON2 [Sem05].

The basic inference on concept expressions in Description Logics is *subsumption*, typically written as $C \sqsubseteq D$. Determining subsumption is the problem of checking whether the concept denoted by D (the *subsumer*) is considered more general than the one denoted by C (the *subsumee*). In other words, subsumption checks whether the first concept always denotes a subset of the set denoted by the second one. For example, one might be interested in knowing whether $\text{Old_Building} \sqsubseteq \text{Tourist_Attraction}$. This kind of the relationship is verified based on the relationships defined in the terminology. Another typical inference on concept expressions is concept *satisfiability*, which is the problem of checking whether a concept expression does not necessarily denote the empty concept. In fact, concept satisfiability is a special case of subsumption, with the subsumer being the empty concept, meaning that a concept is not satisfiable.

In DLs, a distinction is drawn between the *TBox* (terminological box) and the *ABox* (assertional box). In general, the TBox contains sentences describing concept hierarchies (i.e., relationships between concepts) while the ABox contains *ground* sentences stating where in the hierarchy individuals belong (i.e., relationships between individuals and concepts). For example, the statement "(1) *Every employee is a person*" belongs in the TBox; while the statement "(2) *Bob is an employee*" belongs in the ABox. Note that logically, the TBox/ABox distinction is of no significance in the sense that in first-order logic (which subsumes most DLs), the two "kinds" of sentences are not treated differently. When translated into first-order logic, a subsumption axiom like (1) is simply a conditional restricted to unary predicates (concepts) with only variables appearing in it, a sentence of this form is not privileged or special over sentences in which only constants ("grounded" values) appear like (2).

The primary reason for the distinct representation of the ABox and TBox is that the separation can be useful when describing and formulating decision procedures for

various DLs. For example, a reasoner might process the TBox and ABox separately, in part because certain key inference problems are tied to one but not the other, e.g., *classification* is related to the TBox, *instance checking* to the ABox. Another example is that the complexity of the TBox can greatly affect the performance of a given decision procedure for a certain DL, independently of the ABox, and thus having a way to talk about that specific part of the knowledge base is useful. The secondary reason is that the distinction can make sense from the knowledge base modeler's perspective. It is plausible to distinguish between the terms/concepts in the real world, i.e., class axioms in the TBox, and particular manifestations of those terms/concepts, i.e., instance assertions in the ABox.

2.4.2 Description Logics

The basic elements in a description logic are primitive concepts, primitive roles, the universal concept \top and the bottom concept \perp . A *concept* is an unary predicate ranging over the domain of individuals. The *meaning* of a concept is the set of individuals. Complex concepts and roles can be built from primitive ones using available description logic constructors. The terminology defines relevant concepts of the domain and their properties. Then individuals occurring in the domain are described using this terminology [BCM⁺03].

\mathcal{ALC} is a minimal Description Logic including full negation and disjunction, i.e., it is propositionally closed. Basic description logic constructors or syntax, as found in \mathcal{ALC} [BH91] are: $\neg C$ (negation), $C \sqcap D$ (conjunction), $\forall R.C$ (value restriction) and $\exists R.\top$ (limited existential quantification) where C and D are concepts and R is a role. In order to define the semantics of the \mathcal{ALC} concepts we consider the interpretations \mathcal{I} that consist of a non-empty set $\Delta^{\mathcal{I}}$ (the domain of the interpretation) and an interpretation function, which assigns to every concept C a set $C^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$ and to every atomic role R a binary relation $R^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$. The interpretation function is extended to concept descriptions by the following inductive definitions:

$$\begin{aligned} \top^{\mathcal{I}} &= \Delta^{\mathcal{I}} \\ \perp^{\mathcal{I}} &= \emptyset \\ (\neg C)^{\mathcal{I}} &= \Delta^{\mathcal{I}} \setminus C^{\mathcal{I}} \\ (C \sqcap D)^{\mathcal{I}} &= C^{\mathcal{I}} \cap D^{\mathcal{I}} \\ (\exists R.C)^{\mathcal{I}} &= \{x \in \Delta^{\mathcal{I}} \mid \exists y.(x, y) \in R^{\mathcal{I}} \wedge y \in C^{\mathcal{I}}\} \\ (\forall R.C)^{\mathcal{I}} &= \{x \in \Delta^{\mathcal{I}} \mid \forall y.(x, y) \in R^{\mathcal{I}} \rightarrow y \in C^{\mathcal{I}}\} \end{aligned}$$

The $\mathcal{ALC}_{\mathcal{R}^+}$ [Sat96], also referred as \mathcal{S} description logic is an extension of \mathcal{ALC} with transitive roles. Likewise, the other letters are used for various features of description logics ([BCM⁺03], pp.494-495). The letter \mathcal{H} adds role hierarchies, and the letter \mathcal{I} adds inverse roles. \mathcal{ALCF} extends \mathcal{ALC} with functions. Unqualified number restrictions and nominals are added by \mathcal{N} and \mathcal{O} . The former allows stating that any individual is related to at most (or at least) n individuals by a given role.

This way, the functionality of relations can be specified. Nominals allow for explicitly enumerating the members of a concept. The description logic \mathcal{SHIQ} [HST00] corresponds to the $\mathcal{ALCQHI}_{\mathcal{R}^+}$ logic that extends $\mathcal{ALC}_{\mathcal{R}^+}$ with inverse roles, role hierarchies and qualified number restrictions ($\geq n R.C$ and $\leq n R.C$). Qualified number restrictions play an important role for representing and for reasoning about conceptual models because they add the ability to model cardinalities of relationships [BCdG01]. The semantics of the above mentioned extensions are as follows:

qualified number restrictions

$$\begin{aligned} (\exists^{\geq n} R.C)^{\mathcal{I}} &= \{x \in \Delta^{\mathcal{I}} \mid \#\{y \mid (x, y) \in R^{\mathcal{I}} \wedge y \in C^{\mathcal{I}}\} \geq n\} \\ (\exists^{\leq n} R.C)^{\mathcal{I}} &= \{x \in \Delta^{\mathcal{I}} \mid \#\{y \mid (x, y) \in R^{\mathcal{I}} \wedge y \in C^{\mathcal{I}}\} \leq n\} \end{aligned}$$

inverse roles

$$(R^-)^{\mathcal{I}} = \{(x, y) \in \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}} \mid (y, x) \in R^{\mathcal{I}}\}$$

role transitivity axiom

$$(R^+)^{\mathcal{I}} = \bigcup_{i \geq 1} (R^{\mathcal{I}})^i$$

roles hierarchy axiom

$$(R \sqsubseteq S)^{\mathcal{I}} = R^{\mathcal{I}} \subseteq S^{\mathcal{I}}$$

Description logics described above are capable to represent only binary roles, whereas in many real world situations, a role is required to link several concepts. An extension to the description logics with n -ary roles is proposed in [CdGL97]. The description logic \mathcal{DLR} is a proper generalization of the \mathcal{ALCQI} description logic. The basic technique used in \mathcal{DLR} , is *reification* which allows the reduction of the logical implications in \mathcal{DLR} to logical implications in \mathcal{ALCQI} [BCM⁺03].

Another type of the DL extension which is relevant to our work is the ability to deal with complex or composed roles such an extension is described in [HLM99], and concrete domains. Extending descriptions logics with concrete domains is a way to introduce new data types such as integer or rational, or to deal with specific dimensions of objects such as spatial or temporal features. Datatypes are added by the suffix (\mathcal{D}). The $\mathcal{ALC}(\mathcal{D})$ [BH91] description logic extends the \mathcal{ALC} DL by concrete domains and predicates on these domains. Some examples of the concrete domains are the integers, the reals, sets of temporal intervals, or sets of spatial regions; predicates include equality, temporal overlapping, and spatial disjointness. Formally, a concrete domain consists of a set such as the natural numbers, and a set of predicates such as binary " $<$ " and the ternary " $+$ ". The definition of a concrete domain given in [BH91] is as follows:

DEFINITION 1 *A concrete domain \mathcal{D} is a pair $(\Delta_{\mathcal{D}}, \Phi_{\mathcal{D}})$, where $\Delta_{\mathcal{D}}$ is a set and $\Phi_{\mathcal{D}}$ is a set of predicate names. Each predicate name $P \in \Phi_{\mathcal{D}}$ is associated with an arity n and n -ary predicate $P^{\mathcal{D}} \subseteq \Delta_{\mathcal{D}}^n$.*

To illustrate DEFINITION 1, let us present the definition of the *numerical* domain \mathcal{Q} (from [Lut03]). The set $\Delta_{\mathcal{Q}}$ is the rational numbers \mathbb{Q} ; the set of predicates $\Phi_{\mathcal{Q}}$ is:

- unary predicates $P_{\mathcal{Q}}$ for each $P \in \{<, \leq, =, \neq, \geq, >\}$ and each $q \in \mathbb{Q}$ with $(P_q)^{\mathcal{Q}} = \{q' \in \mathbb{Q} \mid q' P q\}$;

- binary comparison predicates $<$, \leq , $=$, \neq , \geq and $>$;
- ternary predicates $+$ and \mp with $(+)^{\mathcal{Q}} = \{(q, q', q'') \in \mathbb{Q}^3 \mid q + q' = q''\}$ and $(\mp)^{\mathcal{Q}} = \mathbb{Q}^3 \setminus (+)^{\mathcal{Q}}$;
- a unary predicate $\top_{\mathcal{Q}}$ with $(\top_{\mathcal{Q}})^{\mathcal{Q}} = \mathbb{Q}$ and a unary predicate $\perp_{\mathcal{Q}}$ with $(\perp_{\mathcal{Q}})^{\mathcal{Q}} = \emptyset$.

An approach to extend concept languages (preceding name for description logics) with concrete domains was proposed in [BH91]. The main properties of such an extension of a concept language as stated in [BH91] are:

- the extension should have a formal declarative semantics which is as close as possible to the usual semantics employed for concept languages;
- it should be possible to combine the existing inference algorithms for concept languages with well-known reasoning algorithms in the concrete domain in order to get the appropriate algorithms for the extension;
- the approach should provide for extending concept languages by various concrete domains rather than constructing a single ad hoc extension for a specific concrete domain. The formal semantics as well as the combination of the algorithms should already be treated on this schema level.

In [BH91], authors describe the combination of the \mathcal{ALC} language with concrete domains. One of the concrete domains discussed in the paper is based on the Allen's interval calculus [AKPT91] as concrete domain \mathcal{AL} . The $\Delta_{\mathcal{AL}}$ consists of intervals, and the $\Phi_{\mathcal{AL}}$ is built from Allen's basic interval relations with the help of logical connections.

Description logic $\mathcal{ALCRP}(\mathcal{D})$ specifically developed for reasoning about qualitative relations, and topological relations in particular, is described in [HLM99]. Description logic $\mathcal{ALCRP}(\mathcal{D})$ extends $\mathcal{ALC}(\mathcal{D})$ by introducing *defined roles* that are based on the *role-forming* predicate operator over concrete roles (for details refer to Section 4.1). The paper proposes restrictedness criterion for the $\mathcal{ALCRP}(\mathcal{D})$ logic to guarantee its decidability. To demonstrate the expressiveness of the $\mathcal{ALCRP}(\mathcal{D})$ logic it is applied to the modeling of the spatial domain with the \mathcal{S}_2 concrete domain. The $\Delta_{\mathcal{S}_2}$ consists of regions and the $\Phi_{\mathcal{S}_2}$ contains the \mathcal{RCC} -8 relationships between regions as defined in [RCC92]. The authors as well sketched the definition of a temporal concrete domain based on Allen's Interval Algebra. The idea of using the same language and to extend it to cope with different aspects of modeled objects by means of concrete domains, is similar to the notion of dimensions in the MADS model that was introduced in Section 2.2.2. In MADS, if an object type is declared to have a temporal extent, than a set of predefined synchronization relationships that this object type can participate in, is associated to it. As noted in [AL04], the paradigm underlying the presentation of temporal conceptual knowledge in a

description logic with temporal concrete domain, is quite different from a 'classical' temporal DL representation, like the $\mathcal{TL}\text{-}\mathcal{ALCF}$ description logic, [AF98]. The latter is better suited to model objects whose properties vary over time; whereas in the former approach, objects are associated with a fixed temporal extent that can be understood as their lifespan during which, all of their properties remain constant. If we now consider qualitative spatial logics, we find that the recent research adopts the principles of the *Spatial Logic based on Regions and Connections* proposed in [RCC92]. The logic proposed by Randell, Cui, and Cohn in [RCC92], is defined for regions as its atomic arguments, and a set of connections now known as $\mathcal{RCC}\text{-}8$ relationships. Essentially, what is defined in the [RCC92] is now referred as the spatial concrete domain. We present in more detail the two domains, spatial and temporal, in the following section.

2.4.3 Description Logics with Concrete Domains.

Spatial domain.

The spatial concrete domain in description logics is based on the Randell's Region Connection Calculus \mathcal{RCC} [RCC92]. The \mathcal{RCC} is an interval logics for reasoning about space. The basic part of the logics assumes a primitive dyadic relation: $C(r_1, r_2)$ reads as ' r_1 connects with r_2 '. It is defined on regions; it is reflexive and symmetric over the defined basic set of the spatial dyadic relationships. The theory also supports a set of functions that define boolean composition of regions, and a set of topological functions that allow for the explicit representation of the interior, the closure, and the exterior.

$\mathcal{RCC}\text{-}8$ contains eight jointly exhaustive and pairwise disjoint base relationships between spatial regions. The $\mathcal{RCC}\text{-}8$ is sufficiently expressive for various application purposes, Geographic Information Systems (GISs) is one of the prominent domains [WZ00]. The $\mathcal{RCC}\text{-}8$ is decidable. The basic relationship set of the $\mathcal{RCC}\text{-}8$ is defined as follows, with r_1 and r_2 being two regions:

- $DC(r_1, r_2)$ - r_1 and r_2 are disconnected;
- $EC(r_1, r_2)$ - r_1 is externally connected to r_2 ;
- $PO(r_1, r_2)$ - r_1 partially overlaps r_2 ;
- $EQ(r_1, r_2)$ - r_1 is identical with r_2 ;
- $TPP(r_1, r_2)$ - r_1 is a tangential proper part of r_2 ;
- $TPP^{-1}(r_1, r_2)$ - r_2 is a tangential proper part of r_1 ;
- $NTPP(r_1, r_2)$ - r_1 is a non-tangential proper part of r_2 ;
- $NTPP^{-1}(r_1, r_2)$ - r_2 is a non-tangential proper part of r_1 ;

The main reasoning task for $\mathcal{RCC-8}$ can be formulated as follows:

”Given a finite set Σ of spatial formulas, decide whether Σ is satisfiable in a topological space, i.e., whether there exists a topological space \mathcal{I} and an assignment α in it such that $\mathcal{I} \models^\alpha \Sigma$.”

[WZ00] shows that the *satisfiability problem* for $\mathcal{RCC-8}$ is decidable even augmented with the following boolean functions: $sum(r_1, r_2)$; Us - universal spatial region; $compl(r_1)$ - complement of r_1 ; $prod(r_1, r_2)$ - intersection of r_1 and r_2 ; $diff(r_1, r_2)$ - difference of r_1 and r_2 .

A transitivity table for the theory is defined as follows. Given a particular theory Σ supporting a set of mutually exhaustive and pairwise disjoint dyadic relationships, three individuals r_1, r_2 , and r_3 and a pair of dyadic relationships R_1 and R_2 selected from Σ such that $R_1(r_1, r_2)$ and $R_2(r_2, r_3)$ the transitive closure $R_3(r_1, r_3)$ represents a disjunction of all possible dyadic relationships holding between r_1 and r_3 in Σ . Figure 2.5 shows the transitions allowed in the $\mathcal{RCC-8}$.

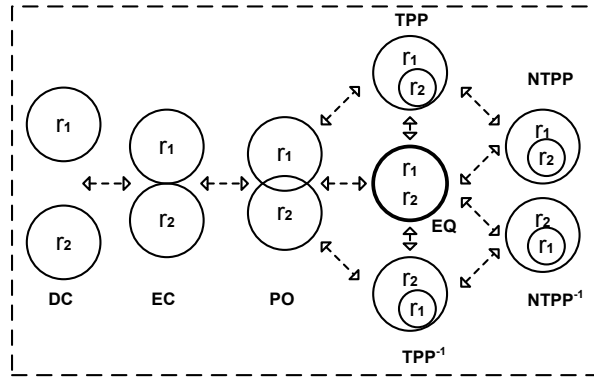


Figure 2.5: Spatial Relationships Transitions.

In [HLM99], authors define spatial domain \mathcal{S}_2 using the $\mathcal{RCC-8}$ relationships as follows:

- unary predicates *is-region* with $is\text{-}region^{\mathcal{S}_2} = \Delta_{\mathcal{S}_2}$ and its negation *is-no-region* with $is\text{-}no\text{-}region^{\mathcal{S}_2} = \emptyset$;
- a binary predicate *inconsistent-s-relation* with $inconsistent\text{-}s\text{-}relation^{\mathcal{S}_2} = \emptyset$;
- the 8 basic predicates *dc*, *ec*, *po*, *tp*, *nttp*, *tpi*, *nttpi* and *eq* that correspond to the $\mathcal{RCC-8}$ relationships and are defined as follows. Let r_1 and r_2 be two regions then, $(r_1, r_2) \in dc^{\mathcal{S}_2}$ iif $(r_1, r_2) \in DC$; $(r_1, r_2) \in ec^{\mathcal{S}_2}$ iif $(r_1, r_2) \in EC$; ...; $(r_1, r_2) \in eq^{\mathcal{S}_2}$ iif $(r_1, r_2) \in EQ$.
- for each distinct set p_1, \dots, p_n of basic predicates, when $n \geq 2$, an additional predicate of arity 2 is defined. The predicate has the name $p_1\text{-}p_2\text{-}\dots\text{-}p_n$ and, $(r_1, r_2) \in p_1\text{-}p_2\text{-}\dots\text{-}p_n^{\mathcal{S}_2}$ iif $(r_1, r_2) \in p_1^{\mathcal{S}_2} \vee (r_1, r_2) \in p_2^{\mathcal{S}_2} \vee \dots \vee (r_1, r_2) \in p_n^{\mathcal{S}_2}$

To form unique predicate names, the following canonical order is imposed - dc, ec, po, tpp, nttp, tppi, ntppi, eq.

For representing and reasoning about spatial objects, spatial description logics have been proposed in the literature. Qualitative spatial reasoning in description logic is based on topological \mathcal{RCC} -8 relationships [CH01],[GN02]. A family of description logics called $\mathcal{ALCI}_{\mathcal{RCC}}$ suitable for qualitative spatial reasoning on various granularity is discussed in [Wes02]. The satisfiability problem of these logics is addressed considering the role axioms derived from the \mathcal{RCC} composition tables. Inverse and disjoint roles are also needed to capture the semantics of these relationships.

Temporal domain.

In temporal description logics, time modeling approaches vary in several ways [AF01]. The unit of time in temporal logics can be either a point or an interval, classifying logics as point-based or interval-based temporal logics. The time can either be implicit or explicit to the logic language. With implicit time, logic formulæ describe event sequences in a state-change style of representation; in explicit logic languages, formulæ are built using temporal operators. Further, with the explicit temporal syntax, a temporal logic can either adopt external or internal point of view on temporal objects. If described externally, a temporal object is associated with a series of "snapshots" in different moments of time, that describe the states of this object in these moments (or intervals). In the case of the internal representation, the language can be seen in a modular way where two different logics are combined, i.e., while an atemporal part describes the static of the objects, the temporal part relates different static object descriptions creating the dynamics in the representation. Description logics with concrete domains follow the internal representation, and they are more suitable to describe properties of temporal objects (e.g., intervals) rather than properties of objects varying in time [AL04].

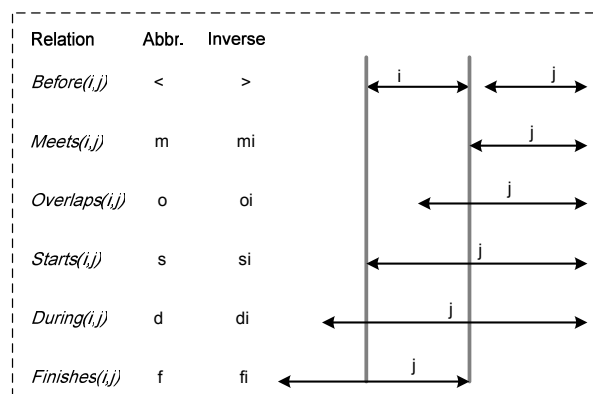


Figure 2.6: Allen's interval relationships.

The concrete temporal domain in description logics is based on the Allen's interval calculus [AKPT91], Figure 2.6 shows interval relationships over some temporal

structure, as defined in [All83]. The most important property of the Allen's interval relationships is that they are jointly exhaustive and pairwise disjoint, i.e., for each temporal structure $(T, <)$ and $(t_1, t_2) \in T$, there exists exactly one relation r such that $t_1 r t_2$. A concrete domain \mathcal{T} can now be defined as follows: In [HLM99], authors define spatial domain \mathcal{S}_2 using the $\mathcal{RCC-8}$ relationships as follows:

- unary predicates **is-interval** with $\text{is-interval}^{\mathcal{T}} = \Delta_{\mathcal{T}}$ and its negation **is-no-interval** with $\text{is-no-interval}^{\mathcal{T}} = \emptyset$;
- a binary predicate **inconsistent-t-relation** with $\text{inconsistent-t-relation}^{\mathcal{T}} = \emptyset$;
- the 13 basic predicates $<$ (before), $>$ (after), **m** (meets), **mi** (met-by), **o** (overlap), **oi** (overlapped-by), **d** (during), **di** (contains), **b** (starts), **bi** (started-by), **f** (finishes), **fi** (finished-by), **=** (equal) that correspond to the Allen's relationships and are defined as follows. Let t_1 and t_2 be two intervals then, $(t_1, t_2) \in <^{\mathcal{T}}$ iff $(t_1, t_2) \in \text{before}$; $(t_1, t_2) \in >^{\mathcal{T}}$ iff $(t_1, t_2) \in \text{after}$; ...; $(t_1, t_2) \in =^{\mathcal{T}}$ iff $(t_1, t_2) \in \text{equal}$.
- for each distinct set $\mathbf{p}_1, \dots, \mathbf{p}_n$ of basic predicates, when $n \geq 2$, an additional predicate of arity 2 is defined. The predicate has the name $\mathbf{p}_1\text{-}\mathbf{p}_2\text{-}\dots\text{-}\mathbf{p}_n$ and, $(t_1, t_2) \in \mathbf{p}_1\text{-}\mathbf{p}_2\text{-}\dots\text{-}\mathbf{p}_n^{\mathcal{T}}$ iff $(t_1, t_2) \in \mathbf{p}_1^{\mathcal{T}} \vee (t_1, t_2) \in \mathbf{p}_2^{\mathcal{T}} \vee \dots \vee (t_1, t_2) \in \mathbf{p}_n^{\mathcal{T}}$. To form unique predicate names, the following canonical order is imposed - $<$, $>$, **m**, **mi**, **o**, **oi**, **d**, **di**, **b**, **bi**, **f**, **fi**, **=**.

A note on spatial and temporal relationships. If we consider the DEFINITION 1 of a concrete domain applied to spatial and temporal concrete domains, we note that the sets $\Delta_{\mathcal{D}}$ of these domains are different, i.e., the set of spatial regions, and the set of intervals respectively; but the set $\Phi_{\mathcal{D}}$ of predicates are similar. Indeed, for each spatial relationship from Figure 2.5 we can find a corresponding interval relationship from Figure 2.6, with the transitions table from Figure 2.5 being valid for the (substituted) temporal intervals as well. Table 2.2 lists these correspondences.

Table 2.2: Spatial and Interval Relationships

Spatial Relationship	Interval Relationship
$DC(r_1, r_2)$	$<(i,j)$ or $>(i,j)$
$EC(r_1, r_2)$	$m(i,j)$ or $mi(i,j)$
$PO(r_1, r_2)$	$o(i,j)$ or $oi(i,j)$
$TPP(r_1, r_2)$	$b(i,j)$ or $f(i,j)$
$TPP^{-1}(r_1, r_2)$	$bi(i,j)$ or $fi(i,j)$
$EQ(r_1, r_2)$	$=(i,j)$
$NTTP(r_1, r_2)$	$d(i,j)$
$NTTP^{-1}(r_1, r_2)$	$di(i,j)$

2.5 Chapter Summary

This thesis advocates a hybrid approach to database schema integration. In this chapter we have presented the data modeling approaches that we employ in our methodology: conceptual modeling and description logic. Conceptual models originate from the database world, provide a clearer representation of the content of the user universe of discourse, and more efficient management for large data sets. DL approaches provide better reasoning capabilities and new knowledge inference from explicitly defined knowledge.

We adopt the mediator-based federated architecture with a global schema that is constructed to meet user requirements for each of the related schemas' elements. We design a methodology for spatial and temporal data that are best described on the conceptual level with an expressive data model. Thus, we employ a conceptual data model that captures spatio-temporal features of the source schemas and allows for their integration through a set of inter-schema mappings. The modeling part of our methodology is supported by the spatio-temporal conceptual model MADS, whereas the validation part of the integration process is delegated to the description logics validation services. This validation phase for spatio-temporal schemas and inter-schema mappings is the original feature of our methodology. We therefore agree that, rather than extending either formalism to try to cover all desirable functionality, a hybrid system, where the database component and the logic component would cooperate, each one performing the tasks for which it is best suited, is a viable solution for semantically rich information management [SPVC04]. Our proposal contributes to the area of research on the following original topics:

- the proposed methodology, based on description logics reasoning mechanisms, conceptual modeling and integrity constraints, is hybrid and thus innovative;
- we are dealing with spatio-temporal data which, to the best of our knowledge, has not yet been adequately covered;
- we are using reasoning mechanisms of description logics in order to validate the set of inter-schema mappings and the source schemas.

We detail our methodology designed for the MADS conceptual model in Chapter 3, and we show how we enhance the methodology with the reasoning services of the description logics in Chapter 4.

Chapter 3

Integration of Spatio-temporal Database Schemas

In this chapter we present our methodology for spatio-temporal database schemas' integration. In our approach we adopt the principles of federated systems where the integration work is done on the schema level leaving the populations untouched. In other words, we propose a methodology to integrate conceptual schemas but not the instances though, we provide the schema integrator with mappings that allow for instance matching. We have developed our methodology for the spatio-temporal data model MADS (Modeling of Application Data with Spatio-temporal features, [PSZ06]), and can it can be considered as an additional feature for modeling spatial and temporal data in the MADS data model. Our integration methodology provides for a flexible integration process with several ways to construct a final integrated schema thus, it results in a schema tuned for the application needs.

As described in Section 2.3, there are three essential types of conflicts to be resolved during the integration: syntactic, semantic, and structural conflicts. For each type, there are several possible ways to resolve the conflicts. When a methodology for integration is proposed it should include solutions for all the types of conflicts. Table 3.1 lists the integration phases, conflicts and the approaches that we have chosen to reconcile the differences in the databases schemas. Following the three-step integration process described in the literature, we realized that we could propose definite, application independent solutions only for the first two phases. For syntactic conflicts, i.e., when data involved in the integration process are logically designed with different approaches, e.g., relational, object-oriented, we propose to remodel the application in MADS conceptual model. For semantic conflicts, i.e., when the same real world facts serve different purposes for different disciplines and they have nothing in common or when the same objects are called differently in different disciplines, resolution in our methodology is done by establishing inter-schema mappings or Inter-schema Correspondence Assertions (ICAs). In more details ICAs are presented in Section 3.4.2. Structural policy taken during the third phase is dependent on the application goals and on the designer perception of the result of

Table 3.1: Integration Phases

<i>Integration phases</i>	<i>Conflicts</i>	<i>Methods</i>
Pre-integration	syntactic, structural	Modeling in MADS
Inter-schema mappings	semantic	MADS correspondence language
Choosing a structural policy	structural	Schematic patterns and integrity constraints
Integrated schema composition		MADS XML language

integration. In addition, different structural patterns have their merits and shortcomings which might not be clear to the designer. Therefore, there is a need of an intermediate step in which the designer is guided through different possible structural policies. Generally, having two entity types **A** and **B** independently of the semantic correspondences existing between them, the resulting schematic solutions can be different, thus, adding another dimension to the set of the decisions to be taken during the integration process. In Section 3.4.3 we present in more details this intermediate phase called '*Choosing a structural policy*'. The results of this phase allow to resolve structural conflicts before designing an integrated schema.

3.1 The Context of the Methodology: Terms and Techniques

For what cases this methodology is applicable. The methodology could be applied in a wide range of situations. The scenario where all the phases of the methodology are maximally exploited is when the same set of real-world objects is modeled from different perspectives and there is a need to combine these different representations; in other words if the populations of the schemas are equivalent and the goal is to provide a common view on their schemas giving access to some or the whole information contained in the source populations. However, the methodology is not limited to the situations where there are common instances in the schema populations; all types of inter-schema mappings can be stated and used in the integration process in the case of disjoint populations.

The methodology is designed for complex domains such as spatial and temporal; in consequence, the syntax of the inter-schema mappings includes spatial and temporal operators and the reasoning services are tuned for such type of data. The methodology is applicable for the thematic data as well, i.e., the data that do not have spatial or temporal features.

Which data model to choose. For our methodology we have chosen a very expressive conceptual data model. To minimize the need for computer aided support at the phase of the discovery of the inter-schema mappings, we base the very first step of the methodology on a model that is semantically powerful, and thus, easily understood by the schema designers. In our methodology we assume that the designer possesses some knowledge in the application domain, but we do not demand our designer to be a domain expert.

Where the process is manual, where automated. Among the integration phases shown in Table 3.1 there are manual, semi-automated and automated. For the very first phase we assume that the conceptual schemas are already translated into the MADS model and we do not consider this translation as part of the methodology. It could be done manually or translated by an engine that provides for such a service. Then, the initial inter-schema mappings are stated manually. The rest of the inter-schema correspondences is proposed for the evaluation for the designer according to the algorithm presented in Section 3.4.3. Once the designer stated all the mappings, this set of mappings together with the schemas are translated into DL by a tool. Additional, application specific integrity constraints that had not been included in the MADS schemas are then added manually directly in the DL editor. The validation of the schemas, and the inter-schema mappings is automated and verified by the reasoner included in the DL editor. Utilization of a reasoner also allows for completion of the set of the inter-schema mappings because a DL reasoner deduces all valid subsumption for the model. Integrated solutions that will be proposed for the designer depend on the results of the validation procedure. If the model is not validated then some integrated solutions involving merging of object types will not be possible.

What is the result. As the result of all four integration phases from Table 3.1 the schema designer will be able to construct an integrated schema based on the complete and valid set of the inter-schema mappings. The reasoning service employed in the methodology ensures there is no local integrity constraint that is violated in the global schema. After the validation procedure the designer has the complete and valid set of the inter-schema mappings, and a set of valid integrated solutions for each population correspondence. As one of the advantages of the methodology we underline that the designer is provided with several valid structural patterns and he can choose one that meets the integration application needs.

How to use these results in practice. The direct application of the methodology is in adopting the results in the design of valid integrated schemas. As a by-product we can see the production of the domain ontology. By translating the schemas and the inter-schema mappings into a DL-based language, which in our case is the Ontology Web Language (OWL) supported in the Protégé editor, we create an ontology of the spatio-temporal domain. This ontology can be enlarged

with more applications, i.e., MADS schemas and inter-schema mappings translated into OWL language of Protégé. The schemas themselves and the mappings contain the information that is used for ontology creation. The ontologies created in the course of the integration process of several schemas can further be used as domain ontologies for new spatio-temporal applications.

In the sequel, we first present the common data model MADS, in Section 3.2; four MADS schemas that are used as motivating examples are presented in Section 3.3. The integration methodology is detailed in Section 3.4. Section 3.4.2 describes our assumptions and presents the first level semantic correspondences. In Section 3.4.2 semantic correspondences for domain values are described. In Section 3.4.3, we introduce our approach to integrity constraints formulation, and present an integrity checking algorithm based on the set of semantic correspondences and integrity constraints. Section 3.4.4 presents the final phase of our approach.

3.2 Introduction to the MADS data model

In Chapter 2.2 we have discussed the place and the role of conceptual models in system engineering. In this section we make a detailed introduction to the MADS conceptual model, and present its main features that allow for semantically rich modeling of spatio-temporal data. There are four dimensions along which properties for the modeled domain can be independently added. These dimensions are: structural, spatial, temporal, and representation. In the following sections we describe each of those dimensions and show how four different property types can be added to the same schema elements.

3.2.1 Structural dimension

Structurally, MADS is an object+relationship data model. It allows schema designers to represent basic concepts from extended entity-relationship modeling, e.g., object type, relationship or association type, *IsA* link, attributes, and methods. Figure 3.1 shows MADS structural notation. Object and relationships bear an identity

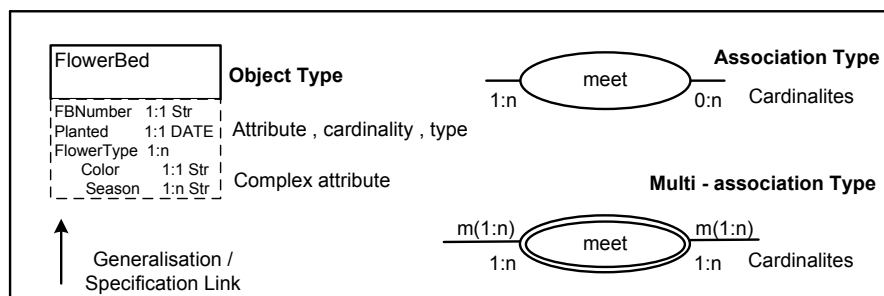


Figure 3.1: MADS structural notation.

and may have attributes. The attributes in MADS are mono-valued or multi-valued, simple or complex, i.e., composed of other attributes. MADS relationship types are n -ary ($n \geq 2$), i.e., they have two or more roles (each role is a link between the relationship type and a given object type), and may be cyclic (two or more roles linking the same object type). Two kinds of relationships provide the basic constructs to link objects together [PSZ05]:

- . **Association.** This is the most common kind of relationship. Roles bear usual (minimum, maximum) cardinalities for the number of relationship instances that object instances of the linked type may participate into.
- . **Multi-association.** This is the most general kind of linking construct. Whereas each role of an association instance links exactly one instance of the linked object type, each role of the multi-association instance links a non-empty set of instances of the linked object type. Consequently, each role bears two pairs of (minimum, maximum) cardinalities. A first pair is the traditional one that defines for each object instance, how many relationship instances it can be linked to via the role. The second, additional, pair defines for each relationship instance, how many object instances it can link with this role. Its minimum cardinality value is at least 1 (in the case when the maximal cardinality is also 1, the multi-association becomes an association).

Example. The following picture shows an example MADS schema where only structural MADS notation is used. The schema has two object types **RoadSection** and **CrossRoad** related through a relationship type **meet**. The natural language explanation of the modeled phenomena is very close to the schematic notation i.e., the modeled objects are road sections, some of them meet crossroads. A road section can begin or end with 0 to n crossroads and a crossroad can be composed of m to n road sections, where $m \geq 2$; this condition is constrained by the cardinalities of the **meet** relationship. \diamond

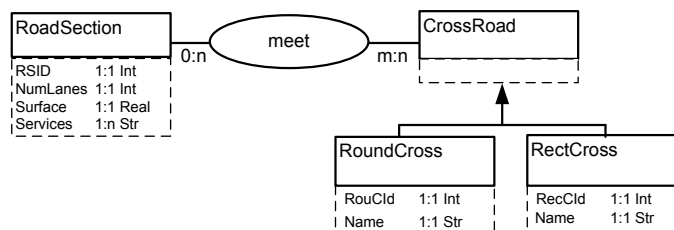


Figure 3.2: MADS dimensions: structural.

3.2.2 Spatial dimension

MADS predefined Spatial ADTs (SADTs) are: point, line, oriented line, simple area, simple geo, point set, line set, oriented line set, complex area, complex geo, geo. The

spatial data types hierarchy is shown in Figure 3.3. Figure 3.3 also shows the icons denoting each SADT. The most generic SADT is `geo`, which generalizes the `simple`-

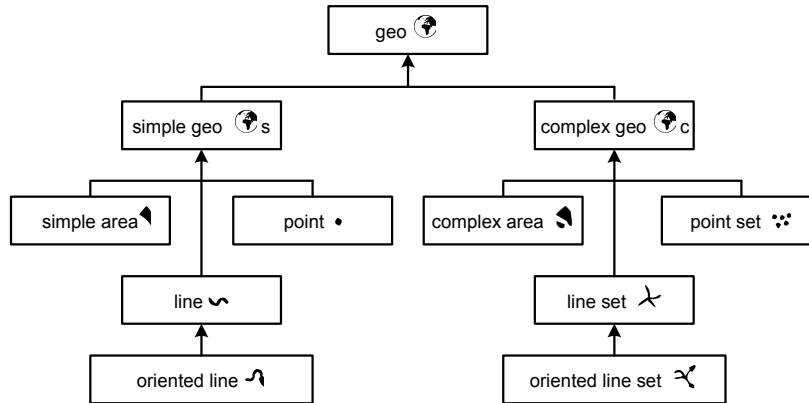


Figure 3.3: MADS basic hierarchy of spatial abstract data types.

`geo` and `complex-geo` SADTs with the semantics: 'this element has a spatial extent' and without any commitment to a specific SADT. These three SADT are abstract and they are never instantiated. The spatiality of an element may either be defined precisely e.g., `point`, `oriented line`, or left undetermined, e.g., `geo`.

Example. The MADS schema in Figure 3.2 can be enriched with the spatial dimension. As Figure 3.4 shows, a road section can be modeled with a geographic domain of the `line` spatial type, a crossroad with the geographic domain of the `geo` spatial type. Consistently with the hierarchy of the SADT, `RoundCross` and `RectCross` are modeled respectively with the `simple area` and `line set` spatial data types.◊

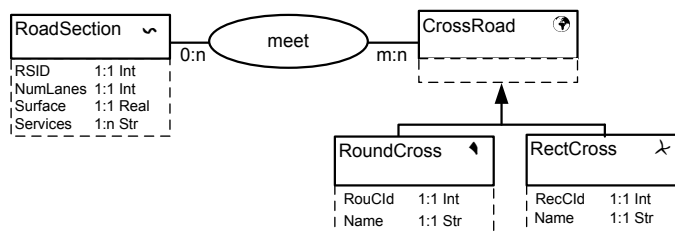


Figure 3.4: MADS dimensions: structural & spatial.

3.2.3 Temporal dimension

Temporal ADTs (TADTs) support timestamping, i.e., associating a time-frame to a fact. Timestamping is the traditional way of modeling temporal information. Timestamped attribute values allow expressing when a value was, is, or will be

holding in the real world as perceived by the application (valid time) or when it was known in the database (transaction time). Timestamped objects and relationships express information on their life cycle: when an object or relationship was created, suspended, reactivated, or deleted. Object and relationship timestamps are also based on either valid time or transaction time. Currently, MADS supports valid time. Figure 3.5 shows the MADS hierarchy of temporal data types.

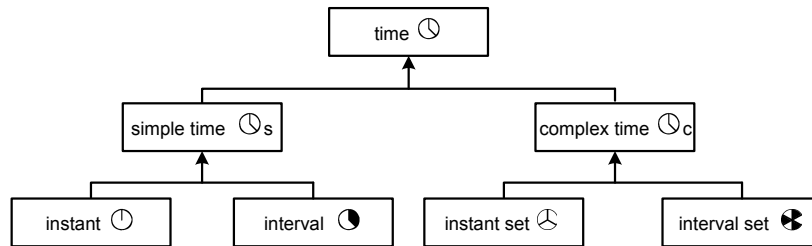


Figure 3.5: MADS basic hierarchy of temporal abstract data types.

Example. We use the same schema as in the previous examples. We can associate temporal properties with the elements of the schema. In the following example the `RoadSection` entity type is modeled as a temporal element, meaning that it has the temporal extent with no commitment to a specific TADT. Its attributes are modeled with the `interval` temporal semantics. \diamond

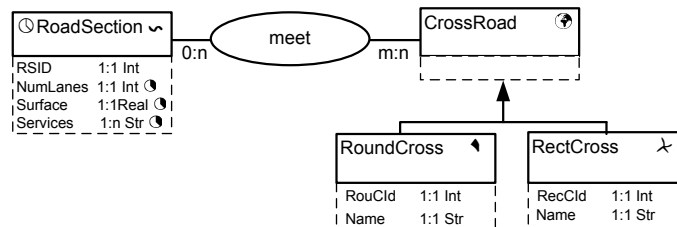


Figure 3.6: MADS dimensions: structural & spatial & temporal.

The spatiality/temporality of an application is reflected by the existence of spatial/temporal entities, but also by the existence of space- and time-related relationships between these entities. It is important to be able to explicitly describe space-related relationships in conceptual schemas. This enriches the schema, allowing these relationships to be named and described with attributes and methods.

3.2.4 Constrained relationships

MADS constrained relationship types are relationship types that convey spatial and temporal constraints on the objects they link. MADS includes topological and synchronization relationships as built-in constrained relationship types. For

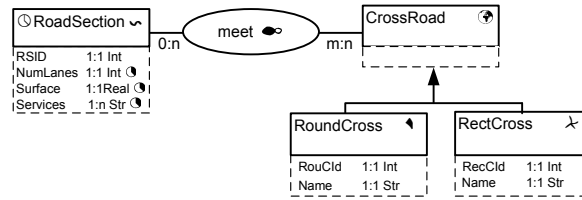


Figure 3.7: MADS dimensions: topological relationship.

example, a topological relationship type **adjacent** may be defined to link object types **CrossRoad** and **RoadSection**, expressing that the geometry of a road section does not intersect but is adjacent to the geometry of a crossroad.¹ The list of MADS predefined topological relationship types and the associated icons is shown in Table 3.2. Every MADS topological relationship type is characterized by its spatial type, which is visually represented by an icon.

Table 3.2: MADS topological relationships.

<i>Spatial type</i>	<i>Icon</i>	<i>Definition</i>
disjunction	○●	the linked objects have spatially disjoint geometries
adjacency	○●	geometry sharing without common interior
crossing	⌘	sharing of some part of interior such that, the dimension of the shared part is strictly inferior to the higher dimension of the linked objects
overlapping	●	sharing of some part of interior such that, the dimension of the shared part is equal to the dimension of the linked objects
inclusion	●	the whole interior of one object is part of the interior of another object
equality	●	sharing of the whole interior and of the whole envelope (for spatial objects of the same dimension)

Example. In Figure 3.7 the **meet** relationship type is enriched with the semantics of the the **adjacency** topological relationship. In this example we cannot add any type of synchronization semantics to the **meet** relationship because of object type **CrossRoad** that has no temporal extent. ◇

Synchronization relationships allow specifying constraints on the life cycles of the participating objects. They allow in particular, to express constraints on schedules

¹The choice of the relationship depends on the application, instead of **adjacent**, **crosses** relationship could be used if it conveys the semantics of the application.

of processes. MADS built-in synchronization relationships are shown in Table 3.3.

Table 3.3: MADS synchronization relationships.

<i>Temporal type</i>	<i>Icon</i>	<i>Temporal type</i>	<i>Icon</i>
equal	\equiv	during	$\dashv\equiv$
meets	$\dashv\lrcorner$	starts	\lrcorner
overlaps	$\dashv\cap$	finishes	$\lrcorner\lrcorner$
before	$\dashv\lrcorner\lrcorner$		

3.2.5 Multiple Representations with Multiple Perceptions in MADS

Multiple representation has been added in MADS as an additional orthogonal dimension. Multiple representation allows the definition in the same schema of several representations for the same real world object. Those multiple representations may be the consequence of diverging requirements during the database design phase or, in the particular context of spatial data, of the description of data at various levels of detail.

To allow users to retrieve specific representations from the set of existing ones, these representations have to be distinguishable and denotable. To this extent, *perception stamps* are added on data, whether they are object type instances or attribute values, meta-data, object or relationship type definitions or attribute definitions. Stamps are vectors of values characterizing the context of each perception, e.g., spatial resolution, viewpoint. Object and relationship types may be perception-varying types and thus have a different set of attributes according to the considered perception.

Example. In Figure 3.8, the object type `RoundCross` is a multi-representation type with two definitions, one for stamp t_1 with the attribute `Roads`, and one for stamp t_2 with the attribute `Name`; attribute `RouCld` exists for both stamps t_1 and t_2 . As we mentioned above, the perception stamps can be added on the level of attributes as well as on the type or relationship levels. As an example, in Figure 3.8 the

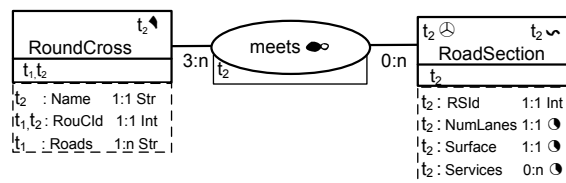


Figure 3.8: Perception varying object type Round Cross.

RoundCross object type has only structural dimension for the stamp t_1 , and becomes a spatial type under the stamp t_2 with **simple area** as its spatial type. For the perception with the stamp t_2 , the **RoundCross** object type is related through a topological relationship **meets** to another spatial object type **RoadSection**. The relationship **meets** in this schema is a constrained relationship, one of the constraints is the spatiality of the related object types. In general, relationship types may hold

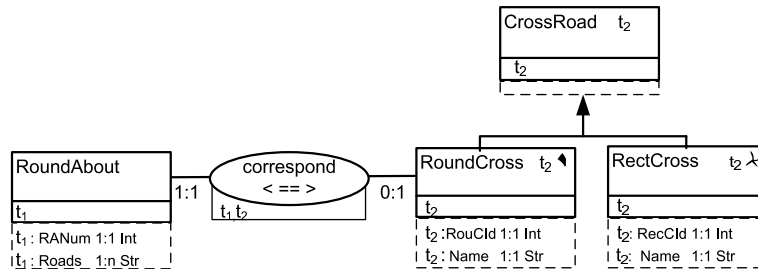


Figure 3.9: Two mono-perception object types linked with the **correspond** inter-representation link.

several different semantics according to the representation and, for instance, be a topological relationship in one representation and a synchronization in another.

There is also a specific inter-representation semantics that may be applied to both associations and multi-associations to denote that the linked object types describe instances that are different representations of the same real world object. As shows Figure 3.9, the same real world objects are modeled with **RoundAbout** object type in a schema with the stamp t_1 and as **RoundCross** object type in a schema with the stamp t_2 . The inter-representation relationship type **correspond** links two mono-perception object types. The cardinality of this relationship states that for each instance from the class **RoundAbout**, there exist exactly one instance of the **RoundCross** type; and there may be at most one instance of the **RoundAbout** type for each instance of the **RoundCross** type. These cardinality constraints of the **correspond** relationship convey the information on how the populations of the related object types are related; with the cardinalities as in Figure 3.9, the population of **RoundAbout** is included in the population of the **RoundCross**. Besides the relationship between the sets of instances, or populations, the inter-representation semantics does not induce any constraints between the linked objects. \diamond

We have introduced the basic elements of the MADS data model, in the sequel we use the structural, spatial, temporal, and multiple representation features to define the syntax and semantics for the inter-schema mappings - a crucial element in our integration methodology. The schema elements that we used to illustrate the MADS dimensions will be further reused in our example schemas.

3.3 Motivating Examples

We use several schemas to illustrate different parts of the integration methodology. First, two schemas S_1 and S_2 are shown in Figures 3.10 and 3.11. We have used parts of these schemas in the previous section to introduce MADS modeling dimensions. Schema S_1 is part of a schema for a park administration of a city. The objects this park administration is interested in are the green plantations within the city area, their geometries, types, e.g., flower bed, park, field. As well there are bordering objects included in the schema, e.g., roundabouts, built-up areas. For these objects, the park administration merely needs to know their names, e.g., roundabout, or their geometry, e.g., road, water body, build-up area. Schema S_2 is a part of a road network schema for road management department of the same city. The focus of this schema is the detailed representation of road network elements, their classifications and the relationships among them. Both schemas model real world elements geographically located in the same area - a city, thus, the populations of these schemas have some common instances providing an integration ground.

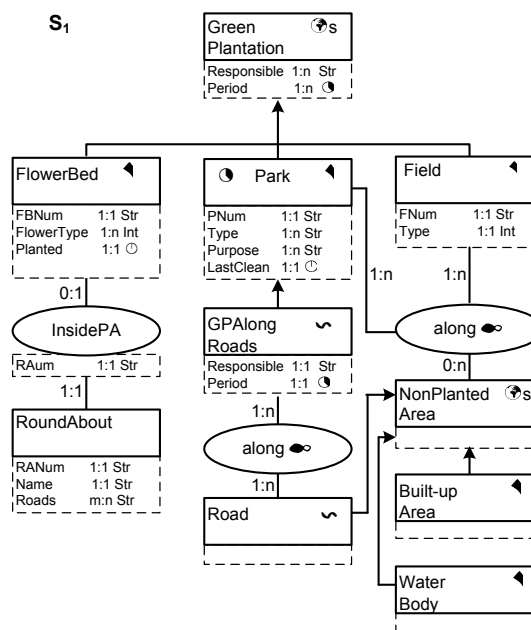


Figure 3.10: Schema S_1 : Park Administration.

There are some concepts used in these schemas which are peculiar to the MADS data model. For example, the *GreenPlantation* object type being non-temporal can have temporal attributes, illustrating the concept of orthogonality of the structural, temporal, and spatial dimensions of MADS data model. *FlowerBed* object type has multi-valued attribute *FlowerType* with 1:n cardinality to illustrate that MADS is an object+relationship data model where attributes with multiple values are allowed. From the point of view of MADS spatial domain ontology, the *IsA* hierarchy of spatial types is coherent.

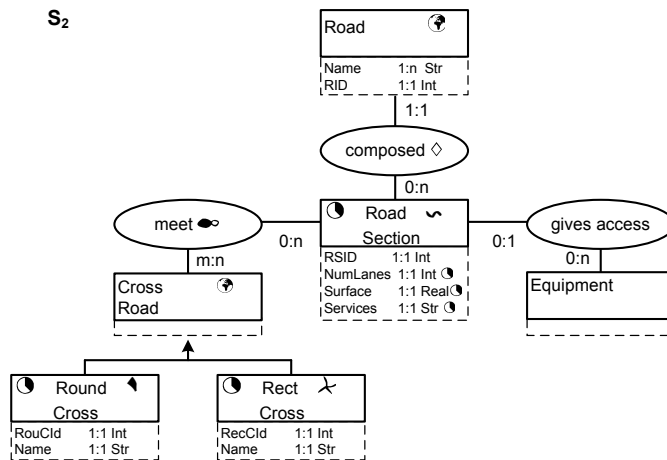


Figure 3.11: Schema S_2 : Road Administration.

Another example that we reference throughout the thesis comprises two database schemas (Figures 3.12 and 3.13) designed by two tourist offices describing the same geographical area, e.g., the city of Paris. The difference in the purposes of these schemas are not as clear as it was for the schemas S_1 and S_2 . The purpose of the schema T_2 is to provide tourists with information on the closest stops of Boats, Bus, Metro, and/or Tram to the tourist sites. Schema T_1 describes the transport means and the tourist sites of the city.

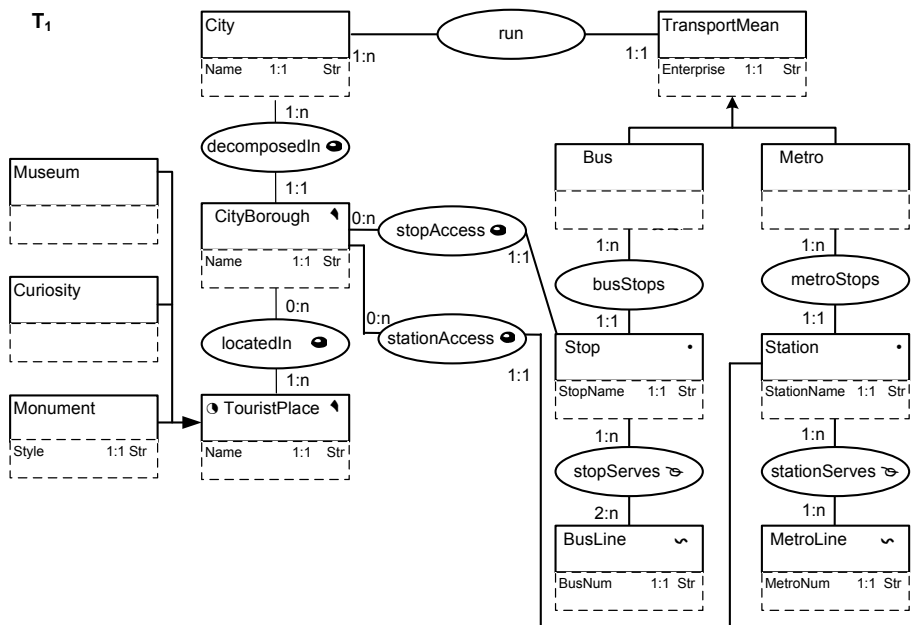


Figure 3.12: Schema T_1 : A City for Tourists.

The schema T_1 models a cadastral division of the city, where the city is decom-

posed in several city boroughs; and some of the urban services available for tourists, e.g., public transport means precisising in which city borough the transport stops are located. The transport means are modeled by the object types **Bus** and **Metro**; the bus and metro stops are modeled as spatial object types **Stop** and **Station**, each **Stop** or **Station** has a spatial extent of type **point**. The city boroughs are also modeled by spatial object types, this allows for a topological relationships **stopAccess** and **stationAccess** between the **Stop** and **CityBorough**, and **Station** and **CityBorough** respectively. These relationships convey the semantics of spatial inclusion, this information is made explicit by the type of the topological relationship. The cardinalities of the **stopAccess** and **stationAccess** relationships state that each transport stop is spatially (or geographically) located in exactly one city borough and that a city borough can have none or several transport stops located in it. Schema T_1 also models information about the geometry of the bus and metro lines. Spatial object types **BusLine** and **MetroLine** have spatial extensions of type **line** which allows storing the coordinates of the bus and metro routes. As it was mentioned in Section 3.2, the spatial and temporal dimensions in the MADS model are orthogonal, meaning that any of the elements of a schema can have one or more dimensions associated with it. In the schema T_1 , object type **TouristPlace** is an example of the spatio-temporal type. It has the spatial extension of type **simple area** and a temporal one of type **interval**. The subtypes of the **TouristPlace** inherit the spatial and temporal properties of their ancestor. The element of the schema that has the largest number of relationships is the **CityBorough** object type, this suggests that this element is the focus of the schema. Rephrasing the latter statement, the focus of the schema T_1 is the description of the city cadastral division; following the relationships of the **CityBorough** object type the user can find the information on what are the transport stops in the given city borough, what are the transport lines crossing it, what are the tourist attractions that can be found in it.

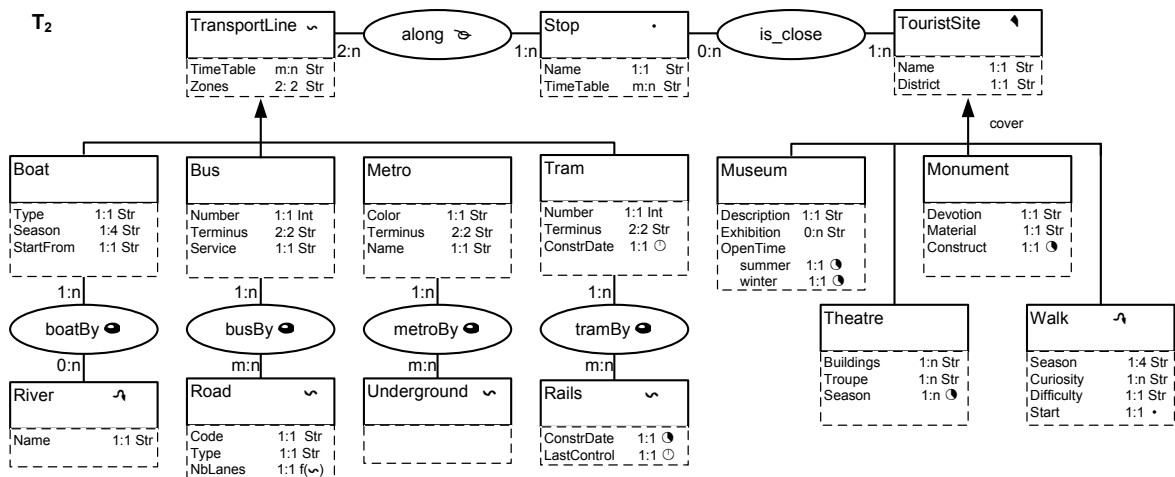


Figure 3.13: Schema T_2 : A City for Tourists.

If we consider the the schema T_2 at a closer look we note that it has a different

focus. Tourist attractions that compose the population of the **TouristSite** object type are those that are accessible by the public transport network. **TouristSite** object type has spatial extension of type **simple area**, **Stop** object type has spatial extension of type **point**. According to the cardinalities of the relationship **is_close**, i.e., 1:n for the **TouristSite** object type, all the instances of the **TouristSite** are located on a certain distance from a transport stop. The subtypes of the **TouristSite** inherit its spatiality except for the **Walk** subtype for which the spatial extent is redefined to the **oriented line** type. In the hierarchy of the **TouristSite** object type the orthogonality of the structural, spatial, and temporal dimensions appears in several object types. The thematic object type **Theatre** has a temporal attribute **Season** of type **IntervalSet**, this attribute specifies the dates of the current theater season. The **Monument** spatial object type with the temporal attribute **Construct** is another example of the combination of the spatiality on the type level with the temporality on the attribute level.

Contrary to the schema T_1 , in the schema T_2 we do not distinguish the types of the stops; what is important is whether a tourist attraction is accessible by any type of public transport. If it is accessible, i.e., if it is in the population of the **TouristSite** object type, then an intuitive user request would follow 'What is the transport means to use to get to this attraction?'. In the schema T_2 , there are four types of public transport means: tram, metro, bus, and boat. In the schema, these four types are modeled as spatial object types with the **line** spatial extension. There are application constraints for the transport line objects. For example, each transport line must have at least two stops along its trajectory, these stops are the terminus stops; each transport line has two zones (that may be the same) associated with it, these zone are identified by the names of the city boroughs where the terminus stops for a given transport line are located. In addition to the transport services, the schema models the means that provide for these transport services. For example, trams run on the rails, the object type **Tram** has a relationship **tramBy** to the object type **Rails**. An instance of **Rails** is a segment of the rail network that provides for a certain tram itinerary, with the geometry of the itinerary included in the geometry of the rail segment. As seen from the above description, the schema T_2 is a tourist-oriented schema, the data are modeled with one of the tourist scenario, where the system user is a tourist, and she/he wants to visit some places accessible by public transport means.

Note that populations of some elements of the schemas T_1 and T_2 are related. For example, the populations of the object types **Museum** of the schema T_2 and **Museum** of the schema T_1 , where the former is included in the latter. Recall that the T_2 models the tourist attractions reachable by the public transport, whereas the T_1 models all the tourist sites, thus, all the museums reachable by the public transport will also be the instances of the more general population of all museums. The same discussion is true for the populations of the **Monument** object types of the two schemas. On the parent level, i.e., **TouristSite** and **TouristPlace**, the populations intersect. The former object type has more subtypes than the latter and therefore,

there are instances of the `TouristSite` object type that do not have corresponding instances in the population of the `TouristPlace`. An example is the `Walk` object type of the schema T_2 .

3.4 Integration Methodology

To recall our integration methodology, we show in Figure 3.14 its four composing phases. We detail in this section each of the integration phases: *Pre-Integration*,

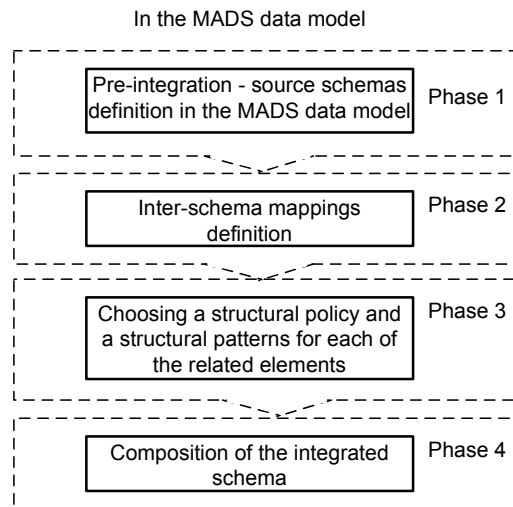


Figure 3.14: Integration Methodology.

Inter-Schema Mappings, *Structural Policy*, and *Integrated Schema Composition*, and illustrate the integration process with the example schemas from Figures 3.10 and 3.11.

3.4.1 Pre-integration

During the pre-integration phase the heterogeneous database schemas are translated into MADS model. In other words, in our approach MADS model is used as the *common* data model [AOTT98] for spatio-temporal data. The translation process itself is out of the scope of our work, but given the existence of the MADS XML schema such a translation should not be resource-consuming. We assume that the application schemas have been already translated to MADS diagrams either with the help of a tool or manually.

3.4.2 Inter-schema Mappings

Complete set of the *Inter-schema Correspondence Assertions* (ICAs) or inter-schema mappings contains following elements:

- relationship between populations of the local or source schemas - *Schema population Correspondences*;
- complete set of the inter-schema mappings between differently represented objects including their identifiers - *Property semantic Correspondences* and *Matching Rules*;
- integrity constraints that together with possible structural patterns are used for validation of a proposed integrated schema.

The set of ICAs consists of inter-schema mappings between populations, domain values and identifiers, and a set of integrity constraints. The mappings are expressed in logical languages, using the following sets: set of schema concept names, denoted as \mathcal{V} ; the set of operators, denoted as \mathcal{O} ; optional set of user-defined functions \mathcal{F} ; an integrity constraint is a formula in a first-order logic language augmented with spatial and temporal predicates.

Schema population Correspondences

Formally, the language for *Schema population Correspondences* (SCs) denoted as \mathcal{L}_{SC} is defined as follows:

$$\mathcal{L}_{SC} = \{\mathcal{V}_{SC}, \mathcal{O}_{SC}\}, \text{ with the syntax}$$

$$[\sigma_{\{\text{SelectExpr}\}}] \text{MCS Operator}_{SC} [\sigma_{\{\text{SelectExpr}\}}] \text{MCS};$$

where \mathcal{V}_{SC} is the subset of \mathcal{V} that contains object and relationship type names but not the attribute names; an MCS or *Modeling Concept Sets* is a subset of \mathcal{V}_{SC} , i.e., the names of object and relationship types that have corresponding types in another local schema. An MCS includes a part of the conceptual schema that models the object involved in the correspondence. An MCS can contain 1 to n elements, it can be either a single entity or relationship type, or a set of connected entity and relationship types. An example of different MCSs where schema elements shown in this figure convey the same information is shown in Figure 3.15. We used different concepts of the data model to express the same information thus, demonstrating that the designer can choose, without any information loss, a representation that meets some design goal, e.g., simplifies further integration process, or emphasizes a crucial element of a schema. We choose MCS as an atomic element of the SC expression because given the expressiveness of MADS model and the entity-relationship approach in general, the same object can equally be modeled by different conceptual primitives, for example, as shown in Figure 3.15. As the atomic element of the correspondence expression, the MCS gives the flexibility to the schema designer to not consider the structural conflicts till the phase where the integrated schema will be composed. The existence of a correspondence between two populations does not

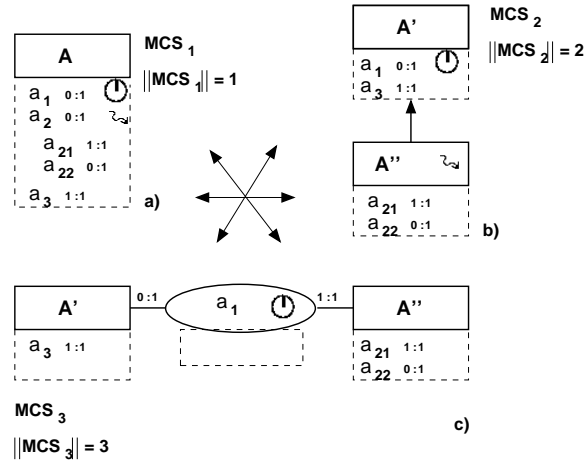


Figure 3.15: Modeling Concept Sets for the same object.

necessarily lead to the merging of the schema fragments that model these populations. The structural patterns that can be applied within the different population relations will be described in the sequel of this section.

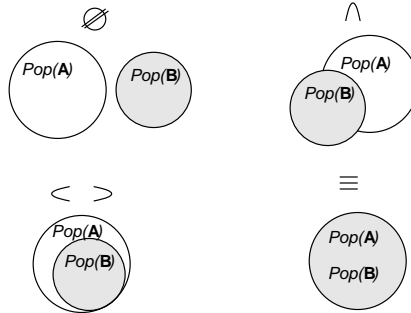


Figure 3.16: Population relationships.

The set of operators for SC expressions denoted as \mathcal{O}_{SC} , contains the following elements: $\{\cap, \subset, \equiv, \emptyset\}$. Graphically, the relationships between two populations can be as shown in Figure 3.16. The Pop(A) and Pop(B) are populations of entity types A and B that are not necessary modeled by a single entity or relationship type, we assume that they are modeled by MCSs.

The choice for the σ operator depends on the possibility to define a condition that selects a subset that correspond to the included (or overlapped) population from the inclusive (or overlapped) one. The usage of an **SelectExpr** refines the inclusion or overlapping relationship and allows a clearer correspondence between the populations. In case of a population correspondence with a σ operator, the relation between the selected populations can be changed to \equiv (equivalence) if the **SelectExpr** is precise enough to exactly select corresponding populations from overlapped ones, or a sub-population of the inclusive one that corresponds to the included one.

The correspondences between the populations are the first expressions to state within our integration process. In some situations the operator chosen for the correspondence is a hypothesis, with the degree of belief depending on the domain knowledge of the integrated schema designer. If none of the operators can be chosen for sure, i.e., the relation between populations is not known, than the disjoint relation should be considered as the initial hypothesis. As it will be demonstrated in the sequel, this choice can be made more precise in the course of the integration process. The disjoint operator describes the weakest degree of the overlapping of the populations, that guarantees that the process of precisising would have the entropy of 1. Already at the level of the SCs, which are the most general correspondences, we can outline possible structural patterns that can be applied to the schema fragments modeling related populations. Putative structural patterns that can be applied to

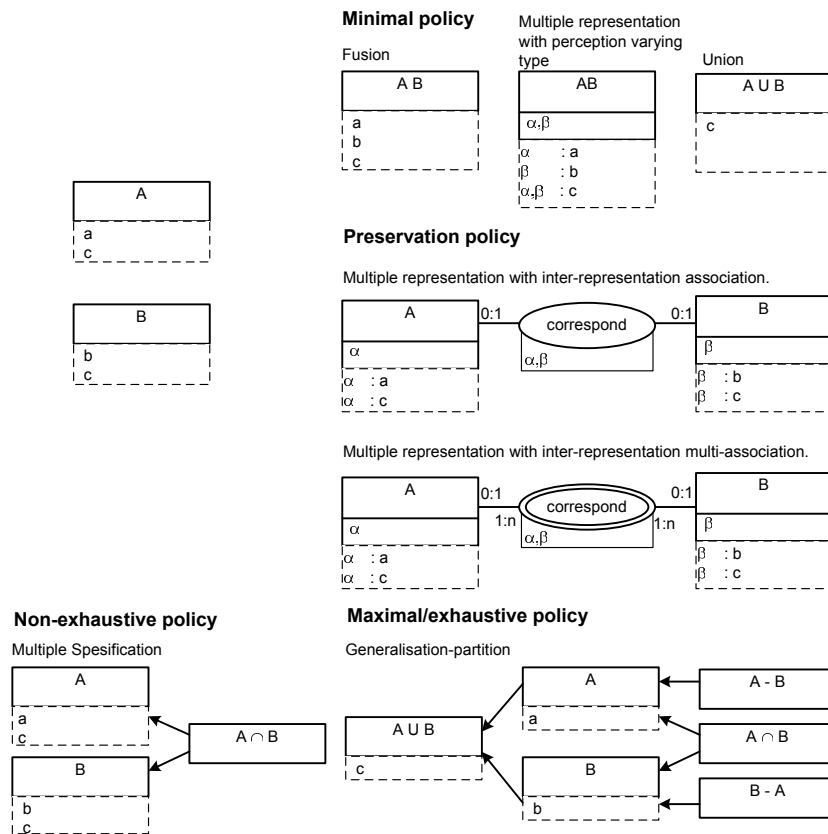


Figure 3.17: Sample integration patterns.

construct an integrated schema are grouped under four policies: minimal, preservation, non-exhaustive, and maximal. These four policies reflect four possible goals that a schema designer may have while constructing the integrated representations for the related schemas' elements. Figure 3.17 shows different structural patterns for two object types A and B with one common attribute c. We assume that there are common instances in both populations: $\text{Pop}(A)$ and $\text{Pop}(B)$.

The *fusion* pattern assumes merging of the populations of both object types **A** and **B** with all their attributes preserved. It means that every instance of the new integrated object type will have attributes coming from both **A** and **B** representations. With the fusion pattern we do not distinguish the origin of the attributes in the resulting integrated object type. To make such a distinction we can apply the attribute stamping by adding representation stamps for each element. In our example we added the α and β stamps to the object types **A** and **B** respectively. The application of this pattern results in a *perception varying* type. With the *union* pattern, only the common attribute **c** is retained in the resulting object type. The population of an object type obtained with the fusion, stamping, or union patterns is the union of two composing populations. The three patterns described above are applied if the integration policy is to propose a minimal structural solution, i.e., a schema with minimal number of elements.

Another integration policy that we call the preservation policy, has the multiple representation with the *inter-representation (multi-)association* [Van01] as its structural solutions. This structural solution preserves the initial representations of object types **A** and **B** and relates them through the inter-representation association or multi-association link. These two types of links differ in the cardinalities of the elements related through them. In the case of an association link, the individual instances are related; in the case of the multi-association link, the related elements are sets of instances. The second set of cardinalities defines the cardinalities of the sets that are related through the multi-association link. A schema element designed with this multiple representation pattern has at least two representations that can differ in their viewpoints and resolutions, where the viewpoint describes user needs, and the resolution specifies the level of details of the representation. These two components define the perception stamp, i.e., perception stamps α and β would have two fields `< viewpoint, resolution >`.

The *non-exhaustive* policy provides the structural patterns for the situation when the designer wants to represent only the representative parts of the populations, i.e., in case of the intersection of the populations (Figure 3.16) there are three representative parts - the intersecting subpopulation and the two sub-populations proper to the **Pop(A)** and **Pop(B)**; in case of the inclusion, there are two representative sub-populations - the intersecting population and the proper **Pop(B)** sub-population. With this policy, the designer would compose an object type for each representative sub-population.

The last integration policy that the schema designer may have in mind is a maximal policy, i.e., to construct a most detailed integrated schema with a large number of object and relationship elements. For such a policy, the designer can apply the *generalization-partition* pattern. With such a structural solution, the populations of both object types are preserved. In addition, the new subtypes of the related populations are instantiated.

The structural patterns shown in Figure 3.17 are applied to the modeling concept sets that are involved in the population correspondences. Schema designer

may decide to apply the same structural pattern within each schema population correspondence and therefore he/she would follow the same policy for all the related elements, or the designer can vary the patterns and possibly the policies for the related elements. To illustrate structural solutions for different SC operators, we will use a rectangle with doubled side lines to depict a MCS, which can be seen as a complex object type. $\text{Pop}(\text{MCS}_1)$ stands for population of a given MCS with a shorter form - Pop_1 . For the examples in the following sections we assume that MCS_1 is equal to the object type A from Figure 3.17, and the MCS_2 is equal to the object type B from the same figure.

Disjoint Populations. An example of the situation when conceptually same objects are modeled by similar schemas but the populations of these schemas are disjoint could be cadastral planning made by different organizations for different cadastral parcels (Figure 3.18). As seen from the figure, the elements to model



Figure 3.18: Cadastral plans.

are the same: roads, buildings, land parcels. We can assume that analyzing those cadastral plans different designers would make similar schemas. This assumption is strengthened by the fact that these two schemas would be designed with the same purpose - cadastral management.

There are several schematic patterns that can be applied to integrate the element of the schemas that belong to disjoint populations. Figure 3.19 shows two possible

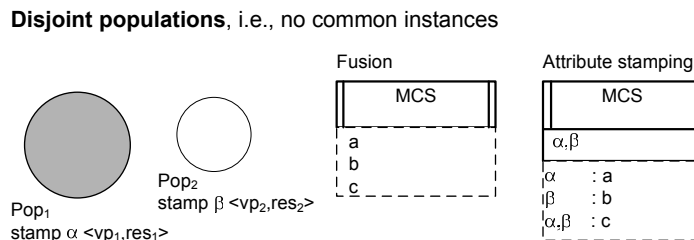


Figure 3.19: Structural patterns for the disjoint operator between the populations of the related modeling concept sets.

structural solutions: fusion and perception varying type. In both cases the two

populations, Pop_1 and Pop_2 are merged, and the object types shown in Figure 3.19 model a population that consists of the Pop_1 and the Pop_2 . In the case of disjoint populations the impact of the application of either of these structural patterns is identical, i.e., we can distinguish between elements originated from Pop_1 or Pop_2 either by a stamp or by a NULL value of one of the attributes. If the value of the attribute a is NULL, then the instance is from Pop_2 similarly, if the value of the attribute b is NULL then the instance is from the Pop_1 . If now we consider the pattern with the attribute stamping, we would follow the same reasoning, i.e., if we are looking for the elements with the stamp α from the Pop_1 , then the attribute b will not be accessible and vice-versa for the elements with the stamp β . Other patterns (cf. Figure 3.17) are not applicable because there are no common instances in two populations.

As we mentioned earlier, the operator for the population correspondence could be a hypothesis and, in the case when the schema designer disposes a weak knowledge about the relationship between the populations, he/she should choose the disjoint operator as the initial hypotheses. This operator can be made more precise, i.e., changed to a more explicit one like overlap, include, or equal, based on the property correspondences and a comparison of the values of key attributes in the related populations. We will show in Section. 3.4.2 what are the conditions for possible operator precision.

Overlapping Populations. As an example of a situation with overlapping populations we could consider our first motivating example (Figures 3.10 and 3.11) with two schemas made by a park administration of a city and by a road network administration of the same city. Figure 3.20 illustrates this example by showing a



Figure 3.20: A fragment of Lausanne city map.

fragment of the city map of Lausanne, where different real world objects are of interest for two different city administrations. The park administration is interested in the green plantations within the city area as well the bordering objects, like roads or built-up areas. The road administration models in details the road network elements as well as adjacent objects. These adjacent objects can be parks, rivers, built-up

areas. Both administrations model real world elements geographically located in the same area - the city, thus, the populations of the schemas designed by these two administrations have some common instances that make schema populations overlapping.

With the overlapping populations, i.e, the \cap sign in the SC expression, the designer may follow any of the policies and may apply any of the patterns shown in Figure 3.17. Figure 3.21 illustrates which patterns should be applied to produce different representation of the integrated population. Following the possible policies,

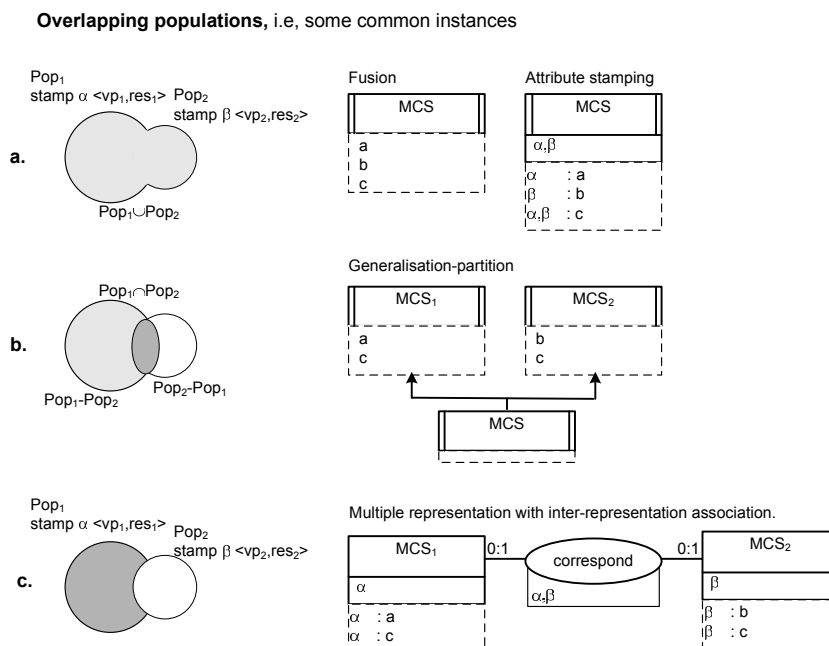


Figure 3.21: Structural patterns for the intersection operator between the populations of related modeling concept sets.

the first solution (Figure 3.21.a) is to produce a minimal schema element as the result of the integration procedure. For this the schema designer would apply the fusion or the attribute stamping pattern. Contrary to the previous situation with disjoint populations, here the result of the application of these patterns is different. Under the fusion pattern, for the instances that belong to both populations we would not be able to keep the information from which population that particular instance originates. With the attribute stamping pattern, this information will be kept in the integrated schema element. Another difference from the previous case with disjoint populations, is that for the overlapping (sub)populations it is necessary to state identical instances and eliminate duplicate values. For this last task the matching rules are employed, we describe the matching rules in Section 3.4.2.

If the designer chooses to adhere to the maximal policy then as shown in Figure 3.21.b, he/she should apply the generalization-partition pattern. This pattern

explicitly defines the population that consists of the instances represented in both populations, and two subtypes that represent the instances belonging only to one of the populations. This pattern exploits the multi-inheritance paradigm of the MADS data model. Similarly to the previous pattern, under the generalization-partition pattern the matching rules are used to delimit the overlapping (sub)populations. We should as well note that in the case of fusion of (sub)populations, e.g., applying the generalization-partition or fusion pattern, the integrity constraint imposed on the source schemas needed to be considered in order to produce a consistent integrated schema. In Section 3.4.3 we present our vision on the role of the integrity constraints in the integration process.

For the last, preservation, policy the structural solution is multiple representation with the inter-representation association with the cardinalities $\{0:1,0:1\}$ (Figure 3.21.c), where the schema elements are stamped with the perception stamps α and β , the association is stamped with both perception stamps. Such a representation allows for keeping the initial populations, it does not require integrity check. Matching rules can be added to such a schema element as auxiliary information to facilitate user queries retrieving identical elements in both populations.

Inclusion of the Populations. An example of the inclusion of the populations could be schemas that model the areas that are geographically included, i.e., a canton and a country that contain this canton, or a district of a town and the whole town. Possible structural solutions in the case of inclusion of the populations are shown in Figure 3.22.

As in the case with disjoint populations, the first two structural patterns are equal (Figure 3.22.a), i.e., with or without stamping we can distinguish the origin of an instance. With the fusion pattern applied, those instances that have all three attributes defined are the instances of both populations, and since the Pop_2 is entirely included in the Pop_1 the former one can be entirely reconstructed from the integrated representation. The generalization-partition pattern (Figure 3.22.b) in the case of inclusion of the populations is reduced to one supertype with one subtype, where the population of the subtype is equal to the the Pop_2 instances with one inherited attribute a from Pop_1 . For last policy, preservation, the structural pattern is shown in Figure 3.22.c. Inclusion of the populations is stated by the cardinalities of the association with the the cardinality for the included population is $\{1:1\}$, and for the inclusive one the cardinality is $\{0:1\}$.

As it is seen from this and previous figures, there are two general approaches: to divide the populations and provide separated models for different representations, or to fuse them. The mining that would be done for a solution with fused populations, either for a complete fusion or with the overlapped population fusion, is the same. By the mining here we mean the discovery of the mappings between the attributes, selection of the key attribute mappings among those, and the validation of the mappings. Once these correspondences are stated and validated the computational time to discover the identical instances by comparing the key attribute values, can

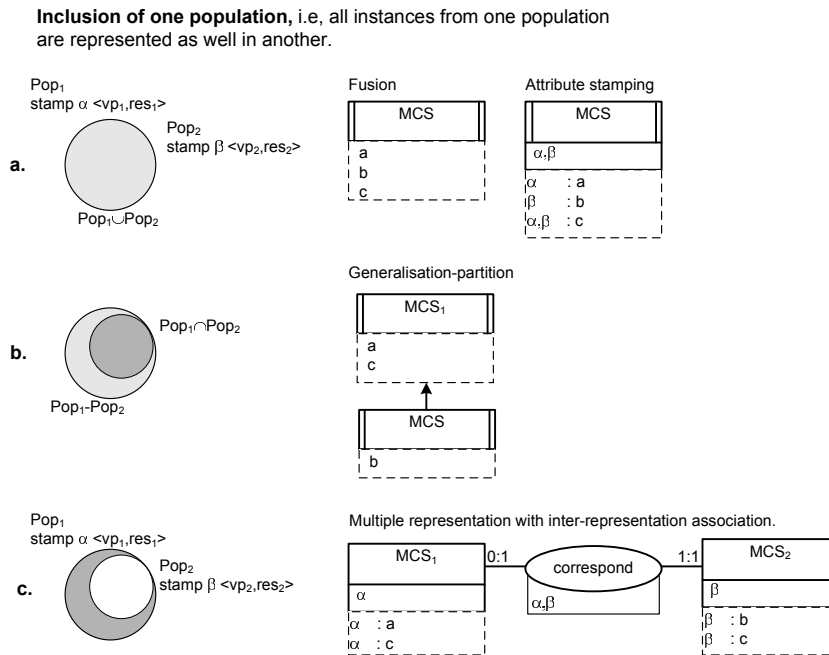


Figure 3.22: Structural patterns for the inclusion operator between the populations of related modeling concept sets.

be neglected. Thus, from the point of view of the complexity and the soundness of the mapping discovery, all patterns with partial or complete fusion require the same analysis.

Equality of the Populations. The equality of the population can be seen as the extreme degree of overlapping of two populations, in comparison with disjointness, intersection, and inclusion of the populations. In the case of the equality of the populations, all instances are represented in both schemas. An example for such a situation could be schemas made by two companies - competitors for a cadastral management system for the same city or country.

Figure 3.23 shows the structural patterns for different policies that are applicable in the case of equal populations. With the equal populations the structural solutions can be total fusion or multiple representation. With the fusion or attribute stamping pattern, in the result representation, all the instances would have all the attribute with non-NULL values. In the multiple representation pattern, the cardinalities of the association become $\{1:1\}$ for both populations. The generalization-partition pattern on the equal populations results in exactly the same representation as already obtained with the fusion pattern.

The set of structural patterns that we use for different policies is not exhaustive, there are more possible patterns defined in [Dup94]. One of such patterns, partition

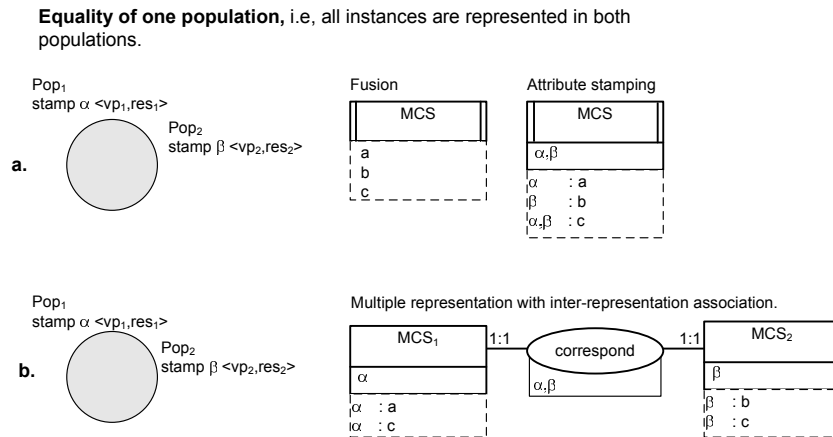


Figure 3.23: Structural patterns for the equality operator between the populations of related modeling concept sets.

with multiple specifications, is shown in Figure 3.24. We had selected the patterns for different policies keeping in mind two goals: to let the schema designer to have all possible representations in the final integrated schema, and not to overburden the choice of the designer. With the structural patterns shown in Figure 3.17, the schema designer can construct any of the patterns listed in [Dup94]. As an example, the structural pattern shown in Figure 3.24, is the generalization-partition pattern without the generalized entity type. Additionally we have classified the patterns by four policies thus narrowing the choice for the designer.

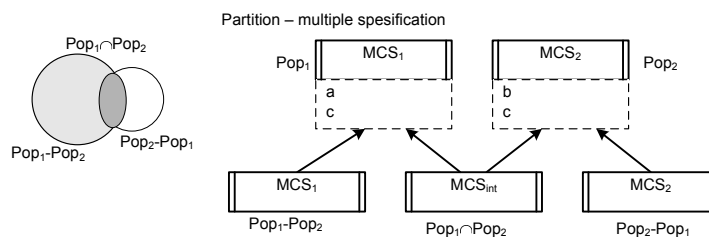


Figure 3.24: Another structural pattern: partition multiple specification.

As it was mentioned above, any solution that contains an entity type or MCS that involves fusion of the populations, even a partial fusion, is equally computationally complex to the total fusion solution which is the simplest from the structural point of view. By structural simplicity we mean the number of conceptual primitives used for modeling, and the cardinality of the set of integrity constraints associated with the representation. The number of mapping rules which defines the computation complexity for a representation, would be the same in both cases, i.e., total or partial fusion. The structural pattern that should be applied depends on the goals of integration and envisioned federated system functionality and data usage. Let us now

consider our motivating example and see what are the population correspondences that exist between the two schemas from Figures 3.10 and 3.11.

Example. Population correspondences between the schemas shown in Figure 3.10 and 3.11 exist between crossroads, e.g., `RoundAbout` and `RoundCross` entity types; and roads modeled as `Roads` entity types. The SCs are the following:

$$S_1.\text{RoundAbout} \subset S_2.\text{RoundCross};$$

$$S_1.\text{Road} \subset S_2.\text{Road};$$

Here we use the \subset operator, because population of `S1.RoundAbout` is only those round crossroads that contain a flower bed inside, whereas population of `S2.RoundCross` is all the round crossroads in the city. The same is true for road sections modeled by the schemas, i.e., in the schema `S1` the population of the object type `Road` are the roads adjacent to some green plantations, whereas in schema `S1` all roads within the city are modeled.

According to the correspondences between the populations, the schema designer has several structural patterns at his disposal, those are the patterns shown in Figure 3.22. Now, for two sets of related populations, the designer should state the attribute correspondences. The structural pattern cannot be chosen based only upon the correspondence between populations, the next essential factor is the integrability of the related populations. In other words, the possibility to state the correspondences between the descriptions (attributes) of the related populations. In the set of ICAs there are two types of correspondences that we call *Property semantic Correspondences* and *Matching Rules* that are meant to assess 'integrability' of the schemas. \diamond

Property semantic Correspondences.

With the *Property semantic Correspondences* the schema designer states the mappings between different representations for same instances. These correspondences are formulated for all the types of the *Schema population Correspondences* including disjoint, because the PCs relate conceptual representations of the objects. The alphabet of the language for the PCs consists of the attribute names of the conceptual primitives that compose the MCSs involved in the SCs, in other words, the PCs unfold the SCs expressions.

The language for property semantic correspondences denoted as \mathcal{L}_{PC} is defined as follows:

$$\mathcal{L}_{PC} = \{\mathcal{V}_{PC}, \mathcal{O}_{PC}, \mathcal{F}_{PC}\}, \text{ with the syntax}$$

[Function]AttributePath Operator [Function]AttributePath;

where \mathcal{V}_{PC} is a set of attribute paths. An attribute path is composed of the name

of a schema, the name of an entity or relationship types, and the attribute name. The set of \mathcal{O}_{PC} includes the mathematical equality sign - = - for attributes having comparable domain type; the sign for a mapping table - \leftrightarrow - this mapping or look-up table is user-defined; the set of spatial relationships - $\circ\bullet$, $\bullet\circ$, \mathcal{D} , \odot , \odot , \bullet - that are used to relate the spatial extensions of the schemas; and the set of temporal relationships - \dashv , \dashv , \dashv , \dashv , \dashv , \dashv , \dashv - to relate the timestamped elements of the schemas.

The set of the *Property semantic Correspondences* is complete if for all the SCs stated for the source schemas, all the pairs of elements (attributes) of MCSs involved, are examined for the existence of a PC between them. In other words, the set of PCs for related populations contains all the attributes that are found/stated as related. Among the PCs there are mappings between the key attributes on one or several MADS dimensions. These key mappings compose the set of matching rules that are described in the next subsection.

Example. For our example schemas there are PCs that are caused by existence of SC, and there are those that are due to overlay of the location or time dependence of the objects modeled by the schemas.

$$\begin{aligned} S_1.\text{RoundAbout.Roads} &= S_2.\text{Road.Name}; \\ S_1.\text{Road} &\bullet S_2.\text{RoadSection}; \\ S_1.\text{FlowerBed} &\odot S_2.\text{RoundCross}; \\ S_1.\text{Park} &\dashv S_2.\text{RoundCross}; \end{aligned}$$

The last two PCs illustrate the situation when there is no population link between two object types but there is a link between spatial and temporal attributes of these entity types. This is an application specific information and it cannot be deduced by any mapping discovery method. Spatial relationship between the *FlowerBed* and the *RoundCross* indicates that a flower bed can lay inside a round about. The condition under which this assertion is true is defined by a corresponding matching rule shown in the example hereafter. The last temporal relationship would correspond to a situation when the road administration for security reasons decides to reconstruct a park surrounded by roads to a round about, as less dangerous then the previous layout or a crossroad. \diamond

Matching Rules.

The *Matching Rules* are the rules that state the correspondences between instances that are represented differently in source schemas. These rules involve identifier attributes.

The language for matching rules denoted as \mathcal{L}_{MR} is defined as follows:

$\mathcal{L}_{MR} = \{\mathcal{V}_{MR}, \mathcal{O}_{MR}, \mathcal{F}_{MR}\}$, with the syntax

[Function]AttributePath Operator [Function]AttributePath;

where \mathcal{V}_{MR} is a set of key attribute paths, where a key attribute can be a primary key or other attribute uniquely identifying instances. \mathcal{O}_{MR} is one of the following equivalence operators including spatial and temporal equivalence $\{=, \leftrightarrow, \bullet, \equiv\}$; \mathcal{F}_{MR} is a set of user-defined functions. On the level of the language alphabet and the set of operators, the \mathcal{L}_{MR} is the subset of the \mathcal{L}_{PC} .

Example. The MRs corresponding to the PCs are the following:

S_1 .RoundAbout.RANum \leftrightarrow S_2 .RoundCross.RouCId;
 S_1 .Road \bullet S_2 .RoadSection;
 S_1 .InsideRA.RANum \leftrightarrow S_2 .RoundCross.RouCId;
 S_1 .Park \bullet S_2 .RoundCross;

For Roads from S_1 and RoadSections from S_2 we do not have any other means for matching instances than to compare their geometry. The last relationship says that if the geometry of a park is equal to that of a round crossroad, then, according to the PCS stated for temporal attributes of these two instances, the park was reconstructed to the round crossroad. \diamond

Till now we have presented the languages for the inter-schema mappings, with those languages the schema designer can state the correspondences between the populations of the related modeling concept sets, the mappings between the attributes including the spatial and temporal extensions of the MADS model. The correspondences stated during this phase of the integration process are thus of two types: extensional and intentional as defined in [CL93]. The extensional correspondences that we employ in our methods are the same as in [CL93], but the set of operators for intentional correspondences is much larger since we provide the mappings between the attributes in three dimensions: structural, spatial, and temporal. The whole set of semantic correspondences is needed to be proven complete so it could be used on the next phase of the integration process. The completeness proof will be demonstrated in Section 5 using the translation of the correspondences in description logics format.

We would like to note that the set of variable names for each type of the correspondences is defined during the pre-integration phase, i.e., the names that are used in the expressions are the object, relationship, and attribute names taken from the MADS schemas. This set of names composes the alphabet of a particular integration process. We can consider this alphabet as constant since we do not add new names but take all of them from the designed schemas. This last remark allows us

to complete the definition of the correspondences language by defining its alphabet, syntax, and semantics.

3.4.3 Choosing a structural policy.

As we have demonstrated in the previous section, structural patterns that can be applied to construct an integrated schema element are conditioned by the type of the SC between the related populations and the integration policy the schema designer chooses to adhere. According to the integration policy, the schema designer has several potential choices for the structural solution for the integrated representation of the related modeling concept sets. There are two composite operations that need to be done within the phase of choosing the applicable structural pattern. The first one is to choose a structural pattern from the possible ones; and the second is to verify that the schema element integrity holds under this structural pattern. Application of a structural pattern entails structural transformation of the initial representations therefore, the integrity of the resulting representation must be verified before this structural pattern will be proposed for the designer as a valid one. The question to be answered is, whether this transformation would lead to violation of the integrity constraints imposed on one or several schemas' elements. The algorithm presented in this section checks the integrity for each schema element produced with a potential structural pattern. If the planned structural transformation is not valid for the given integrity constraints, then the integrity constraints are weakened if it is coherent with the integrated schema usage, or another structural solution is proposed, and the check is run again.

Integrity Constraints.

The integrity constraints represent the rules of the application domain, thus composing an integral part of the conceptual representation of the application data. A conceptual data model that includes the ICs notion allows for capturing and preserving of an additional significant part of the application semantics.

An integrity constraint (IC) is a phrase in a logical language with the alphabet supplied by the schema element names. The schemas supply the following compounds of the ICs:

- *Constant symbols* - object and association type names, attribute and method names, domain types, e.g., `Road`, `FlowerType`, `TimeStamped`. We consider these symbols as constant because in the ICs for a given schema only these and no other names are used.
- *Function symbols* that are expressed as `AttName(EntType)`. For mono-valued attributes, a function results in a variable from the attribute domain, for multi-valued attributes, the result is a set of values, e.g., `Geometry(Road) = Line`, `FlowerType(FlowerBed) = {5,4,3}`.

- *Predicate symbols* that are expressed as $\text{RelName}(\text{EntType1}, \text{EntType2})$, or $\text{MADSPredicate}(\text{EntType1}, \text{EntType2})$, e.g., $\text{Meet}(\text{CrossRoad}, \text{RoadSection})$.

Following is an example with object and association types from Figure 3.10, of an integrity constraint that cannot be expressed with the MADS primitives, and thus requires some additional formalism to be added to the data model. The constraints are expressed in three languages:

- in the natural language: *if the area of a flower bed is more than 5 m², then some of the flowers planted in it should be white roses with corresponding value of the attribute $\text{FlowerType} = 3$;*
- in a logic language: $\forall x, x \in \text{Pop}(\text{FlowerBed}) \wedge \text{Area}(x) > 5 \rightarrow 3 \in \text{FlowerType}(x)$;
- in two algebras:
 - $\text{selection}[(\text{Area} > 5) \wedge (\text{selection}[\text{FlowerType} = 3] \text{FlowerType} \neq \emptyset)] \text{FlowerBed} = \emptyset$;
 - $\text{selection}[(\text{Area} > 5) \wedge \neg \exists t \in \text{FlowerType}(t = 3)] \text{FlowerBed} = \emptyset$;

The compatibility of ICs is analyzed if the object types under constraints are involved in a SC and according to the chosen structural policy and pattern the schema designer decides to fuse (totally or partially) the populations of the source schemas. The component ICs must be studied to deduce a common, *global* ICs guaranteeing its consistency.

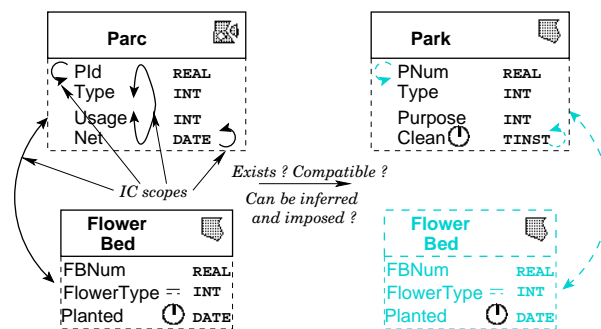


Figure 3.25: IC compatibility verification.

Graphical representation of the compatibility procedure is shown in Figure 3.25, this figure illustrates the validity check algorithm that is presented in the following subsection. Let us assume that there are two object types Parc and Park with equal populations. Following a minimal integration policy, the schema designer had chosen to construct an integrated object type by applying the fusion pattern (cf. Figure 3.17). Each object type has its integrity constraints. Let us assume that there is an

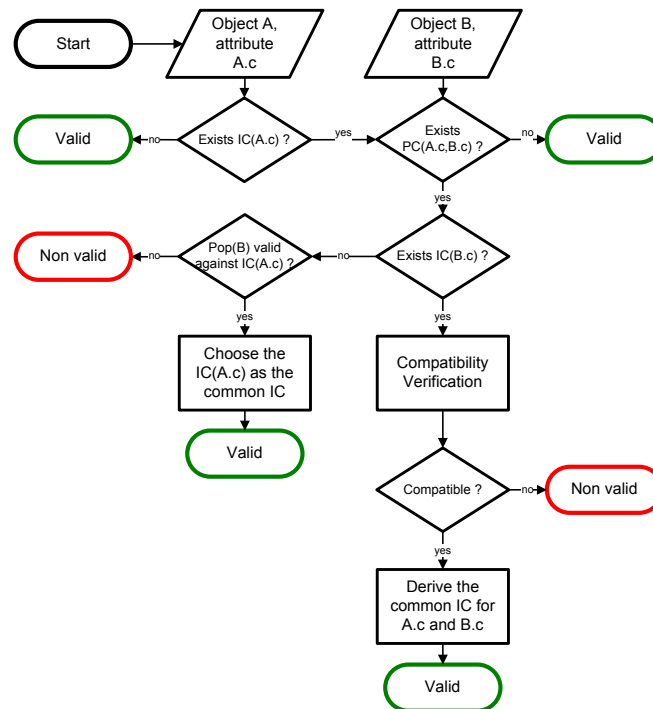


Figure 3.26: Validation block diagram for two object types A and B with a common attribute c.

integrity constraint imposed on the `Parc.Type` attribute. Then, the integrity check procedure would first look whether there is a property semantic correspondence defined for the same attribute; if it exists, for example $PC(\text{Parc.Type}, \text{Park.Type})$ then the next check is whether there is an integrity constraint imposed on the `Park.Type`. If such a constraint exists, i.e., $IC(\text{Park.Type})$, then the compatibility check on $IC(\text{Parc.Type})$ and $IC(\text{Park.Type})$ is run. If two integrity constraints are found compatible, then the common integrity constraint is derived based on the initial constraints. If they are not compatible, then the designer should choose another pattern. If, on the other hand, there is an attribute of `Park` object type that is involved in a PC, but there is no integrity constraint imposed on it, the validity procedure will check whether the integrity constraint $IC(\text{Parc.Type})$ could be imposed on the `Park.Type` attribute and therefore, be proposed as the common integrity constraint for the integrated representation of two attributes. The same analysis is done for integrity constraints of all types, i.e, domain constraints, constraints between several attributes of the same object types, and constraints between the attributes of different object types.

A block diagram in Figure 3.26 shows the validation procedure for two attributes `A.c` and `B.c`, we always use the same object types A and B from Figure 3.17. Such a procedure is run for all pairs of the attributes of the object types involved in a SC. If the result of all runs is **valid**, then the pattern chosen by the designer is valid,

and can be applied for the integrated representation of the related object types. In the following subsection we present the algorithm which is run for every population correspondence expression. This algorithm insures the completeness and validity of the set of inter-schema mappings defined within a population correspondence. As its part it implements the block diagram shown above.

Compatibility Verification Algorithm.

To finally decide upon the structural pattern to be applied in the integration of related MCSs we need to assess the proposed solution or integrability of the MCSs involved into population correspondences.

We propose the algorithm that uses the set of local integrity constraints and inter-schema mappings to check the schema integrity. This algorithm details the procedure shown in Figure 3.25. We let the following assumptions:

- assume two conceptual schemas S_1 and S_2 with two sets of local integrity constraints IC_{S_1} and IC_{S_2} ;
- S_1 and S_2 are valid against their local integrity constraints;
- the two sets IC_{S_1} and IC_{S_2} are complete;
- in the following algorithm we will use the entity types A and B from Figure 3.17 as elements of schemas S_1 and S_2 respectively;
- we denote the set of *Schema population Correspondences* between MCSs A and B ² as SC_{AB} ; the set of property semantic correspondences as PC_{AB} ; integrity constraints are denoted according to the schema element they invoke, for example, a set of ICs on the attribute c is denoted as IC_c or $IC_c(S_1.A.c)$ to precise the element invoked. The elements of these sets are distinguished by additional indexes, for example an element of the PC_{AB} is denoted as PC_{AB_i} ;
- the set of inter-schema correspondence assertions is complete.

ALGORITHM:

1. the schema designer proposes a population correspondence and a set of inter-schema mappings within it. Based on our assumption of the completeness and validity of the local integrity constraints, the completeness and validity of the population correspondence and the inter-schema mappings are to be checked, go to 2.
2. Completeness check. If there is a correspondence between populations of two entity or relationship types then, all possible pairs of their attributes are proposed to the user as potentially related, go to 3.

²here we use A and B to denote MCSs to simplify the notation of the algorithm.

3. Validity check.

```

3.1 assume exists a  $SC_{AB_1}$  between entity type  $A$  and  $B$  expressed as
     $S_1.A \operatorname{operator}_{SC} S_2.B$ 
3.2 IF the set of  $PC_{AB}$  is empty THEN GOTO 5
3.3 ELSE
3.4 WHILE the set of  $PC_{AB}$  is not empty DO
    assume exists a  $PC_{AB_i}$  on the common for  $A$  and  $B$  attribute  $c$ 
    expressed as  $S_1.A.c \operatorname{operator}_{PC} S_2.B.c$ 
3.5 WHILE the set of  $IC_c$  is not empty DO
    assume that in  $S_1$  there is an  $IC_{c_j}$  on attribute  $c$  expressed as
     $IC_{c_j}(S_1.A.c)$ 
3.6 IF exists an  $IC_{c_k}(S_2.B.c)$  THEN
3.7 IF  $IC_{c_j}(S_1.A.c)$  AND  $IC_{c_k}(S_2.B.c)$  are compatible THEN
    continue (i.e., iterate on  $IC_c$  OR  $PC_{AB}$ )
3.8 ELSE  $PC_{AB_i}$  OR  $SC_{AB_1}$  is wrong, GOTO 1
3.9 ENDIF
3.10 ELSE propose an  $IC_{c_k}(S_2.B.c)$  to the user based on  $SC_{AB_1}$  and
     $PC_{AB_i}$ 
3.11 IF the  $IC_{c_k}(S_2.B.c)$  is valid according to user validation
    THEN continue (i.e., iterate on  $IC_c$  OR  $PC_{AB}$ )
3.12 ELSE GOTO 3.8
3.13 ENDIF ENDIF
3.14 ENDDO ENDDO
3.15 ENDIF
3.16 GOTO 4

```

4. Global integrity constraints IC_{global} . Once the set of mappings is complete and valid, global integrity constraints can be derived based on the sets of local integrity constraints and the mappings.

```

4.1 IF exists  $PC_{AB_i}$  expressed as  $S_1.A.c \operatorname{operator}_{PC} S_2.B.c$  AND
     $IC_{c_j}(S_1.A.c)$  AND  $IC_{c_k}(S_2.B.c)$  THEN
     $IC_{global} = IC_{c_j}(S_1.A.c) \operatorname{operator}_{IC} IC_{c_k}(S_2.B.c)$ 
4.2 ENDIF

```

where, $\operatorname{operator}_{IC}$ depends on the $\operatorname{operator}_{SC}$, $\operatorname{operator}_{PC}$, and the integration policy chosen. Recalling the policies presented in Figure 3.17 under the *Fusion* pattern, the $\operatorname{operator}_{IC}$ is \cup because the set of attributes of the entity type AB is the union of the attributes of A and B ; the *Union* pattern requires the \cap as the $\operatorname{operator}_{IC}$ because the set of attributes of $A \cup B$ is the intersection of those of A and B .

5. For object, relationship types or attributes representing the same real world entities, but modeled with incomparable assumptions, the user is proposed to apply multiple representation pattern.

END ALGORITHM

Considering the implementation of the completeness and validity checks, we will present our approach in Section 5. To implement these checks we employed the description logics reasoning services that provide an inference mechanism to complete a model and validation algorithms. The term model in description logics is equivalent to the term schema in conceptual modeling. To be able to use the description logic services we have implemented the MADS model primitives and the sample schemas as a description logic model. We will detail on this implementation in Section 5.

Example. Using our example and assuming that we need to make an integrated schema based on the two input schemas we can obtain significantly different results. They depend on the purpose of usage of the integrated schema. If the integrated view is created for park administration, we might keep minimal information about the crossroads and the resulting entity type would be modeled as shown in Figure 3.27.a. This entity type is obtained with the fusion technique, with loss of information, i.e., the spatial features are dropped; with loss of precision, i.e., now the cardinality of the link between RoundAbout and FlowerBed is 0:1; and finally with no precise way to reconstruct geometry of crossroads, i.e., geometry can be approximately derived from the geometry of flower beds. But, still, such a representation suits the user needs. If, on the other hand, the integrated schema would be used by

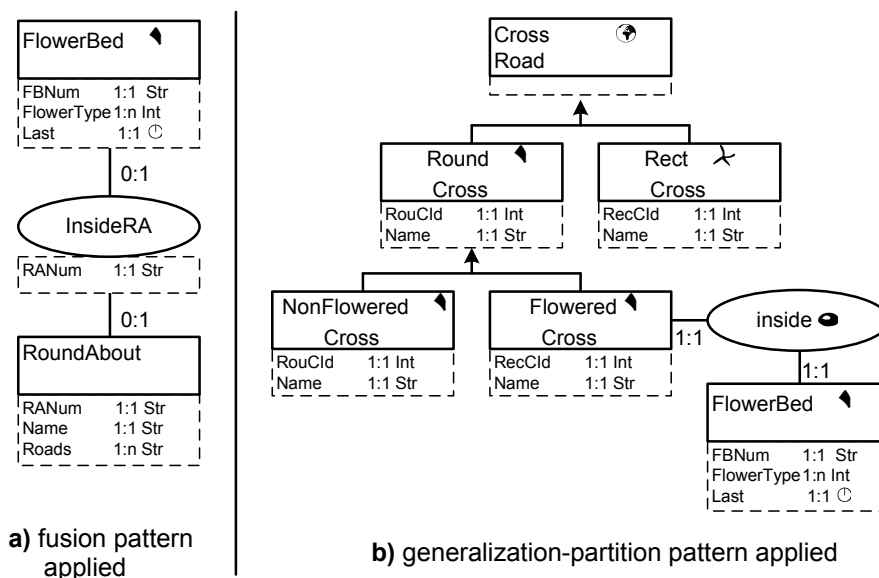


Figure 3.27: Integrated solutions obtained with different patterns.

road administration or for both the divisions, we might want to preserve all the information and maybe enrich a resulting schema with new entity or relationship types.

Figure 3.27.b shows the result obtained with the generalization-partition technique. Regarding the temporal PC between **Park** and **RoundCross**, it can be modeled with additional temporal transition relationship between these entity types. We did not present this type of relationship in the thesis, but it is supported in MADS data model.◊

By the completion of this phase, choosing structural patterns, the designer of the integrated schema has valid structural solutions for the related representations assured by the algorithm above; associated integrity constraints; and mappings for all related elements of the schemas provided by the completeness of the set of the inter-schema mappings. The last phase of the integration method is to compose the integrated schema using the structural patterns chosen for all related schema elements.

3.4.4 Integrated schema composition

The last phase of the integration method is the *Integrated schema composition*. The automatic generation of an integrated schema is feasible if all the following conditions are fulfilled:

- the set of inter-schema mappings is complete;
- the structural patterns are valid against the integrity constraints of the data sources;
- the proposed integrated schema is valid against the new integrity constraints imposed by the structural transformations.

The first condition is ensured upon the completion of the *ICAs formulation* phase, validity of the proposed federated schema is checked on the *Choosing a structural pattern* phase. The steps to generate a final federated schema are:

- to agree on the vocabulary for the integrated schema. Several approaches could be considered, one is to choose a *main* or *trusted* schema among the local schemas and use the names from this schema. Another approach could be to use a domain ontology or a thesaurus to resolve naming heterogeneity. For instance, assume that there is a semantic correspondence:

$$S_1.\text{Green} \equiv S_2.\text{PleasureGround};$$

meaning that the populations of the two entity types **Green** and **PleasureGround** are the same, and according to the schematic transformations these two entity types are to be merged; then the designer could chose one of these names; or having a domain ontology or a thesaurus at hands (Figure 3.28),

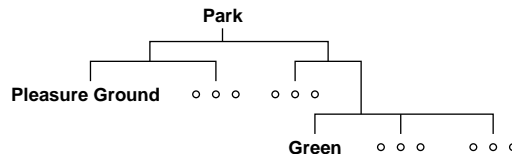


Figure 3.28: A fragment of a domain ontology.

find that these two terms are descendents of the term PARK, and would chose the former to name the integrated entity type;

- to ensure a complementary semantic integrity check, so that the integrated schema is consistent with the expert perception of the application domain. Automatization on this step means that the expert knowledge is formalized and there is a method to compare the schema against the expert knowledge. This can be done by means of the domain ontology, i.e., the generalization and specialization links of the schema should conform the domain ontology hierarchy of terms;
- to formulate the schema in MADS XML and visualize it in the MADS schema editor. There are two XML files to be constructed - a file containing the conceptual representation of the resulting schema, and a file containing screen location of the schema elements for the MADS editor. Both file can be constructed automatically. The conceptual part is constructed based on the results of structural transformations and mappings defined for the related schema elements. The graphical part can be calculated as follows: each element of the schema is assigned a rectangle on the screen of the MADS editor, the size of that rectangle is an average size of a graphical element of the MADS editor. Further, the user of the schema can correct the location of the elements on the screen.

3.5 Chapter Summary

In this chapter we have presented the theoretical foundations for the integration methodology for database schemas extended with spatial, temporal, and representational dimensions. The methodology is based on the conceptual model that features all the above stated dimensions. The core part of our method is the set of inter-schema correspondence assertions that includes population relationships, attribute mappings, and integrity constraints. For all the types of the inter-schema correspondence assertions we have defined the correspondence languages with their alphabets, syntaxes and semantics. The languages exploit the expressive power of the underlying data model and allows for mappings along all its dimensions. The inter-schema mappings can be stated between the schema elements with traditional

domains like integer or character, or with original for our methodology spatial and temporal domains.

Evolving from the relationships between real world sets of related objects our methodology takes into account the relativism of conceptual representation and employs the notion of the multiple concept sets to allow a schema designer an extra flexibility in expressing the inter-schema correspondence assertions. The notion of conceptual relativism embedded in the integration methodology allows for resolving of structural discrepancy in the schemas by means of the data model itself. It partially liberates the schema designer from composing complex correspondence expressions and makes the resulting mappings' set more expressive. The usage of the conceptual relativism notion is coherent with the overall idea of integration because it supports the assumption on the existence of different perceptions of the same real world phenomena.

In our methodology we guide the schema designer towards constructing the integrated schema that corresponds to the application needs. We propose to the designer several structural policies and patterns that result in different integrated schema elements. The designer is not constrained to choose the same policy or pattern for all related populations, instead for each population correspondence he/she can choose the most adapted structural solution. The set of structural patterns defined for the integration process guarantees that there is always at least one patterns that can be applied to produce a valid integrated schema element.

As another original feature, our methodology includes the validation phase where the set of the inter-schema correspondence assertions is checked for the consistency, and the putative structural patterns are validated against the integrity constraints imposed on the source schemas. In this chapter we proposed the validity check algorithm. As the result of the algorithm execution, the schema designer obtains the answer on whether a fusion and therefore an application of a corresponding structural pattern is possible with the given relationship between populations, and the inter-schema correspondence assertions stated in the previous integration phases. The implementation of the completeness and compatibility checks of the algorithm entails the application of a formal reasoning methods, and therefore requires another data modeling approach enhanced with the reasoning services. We present our integration methodology implemented with the validation support in the next chapter. To enhance our integration methodology we have chosen the description logic based data model for its formally-founded reasoning support.

Chapter 4

Validation: Theory

Why validate. In the previous chapter we have described the data model and the integration methodology that we have designed for that model. The methodology features four main phases where 1) source schemas are translated into the MADS model; 2) the inter-schema mappings are expressed in the MADS correspondence language; then, 3) based on the set of the inter-schema mappings and integrity constraints, the structural solutions for related parts of the schemas are proposed to the designer; and finally, 4) the integrated schema is composed. For all of these phases we have proposed the theoretical foundations including the syntax and semantics for the inter-schema mapping languages and, a predefined set of structural patterns for different set relations between source schema populations. The integration process is not automatic: the source schemas are translated into the MADS model by the schema designer, the inter-schema mappings are stated manually, as well as the integrity constraints associated with the source schemas. Thus, our methodology requires a verification mechanism for the totality of the expressions stated manually.

In our methodology we validate the source schemas against the data model; the inter-schema mappings against the semantics of the data model and the syntax of the correspondence language; the compatibility of the integrity constraints imposed on the related populations. In this chapter we present a validation mechanism our methodology is enhanced with. We show how and where the reasoning engine is plugged into our integration methodology to perform validation. This validation phase alters the key points in our integration methodology if compared to other approaches. We shift the emphasis on automation from the a priori discovery to the a posteriori checking of the inter-schema mappings. By doing this, we take advantage of the expressive power of the common data model for the source schemas description and inter-schema mappings definition.

We now show in Figure 4.1 our enhanced integration approach where every manual phase is supported by a validation operation ensuring data model and language compliance. For the phase 1), we aim to verify the validity of the expressions of the source schemas in the MADS model. The MADS data model itself and the source schemas with integrity constraints are translated into a DL based language

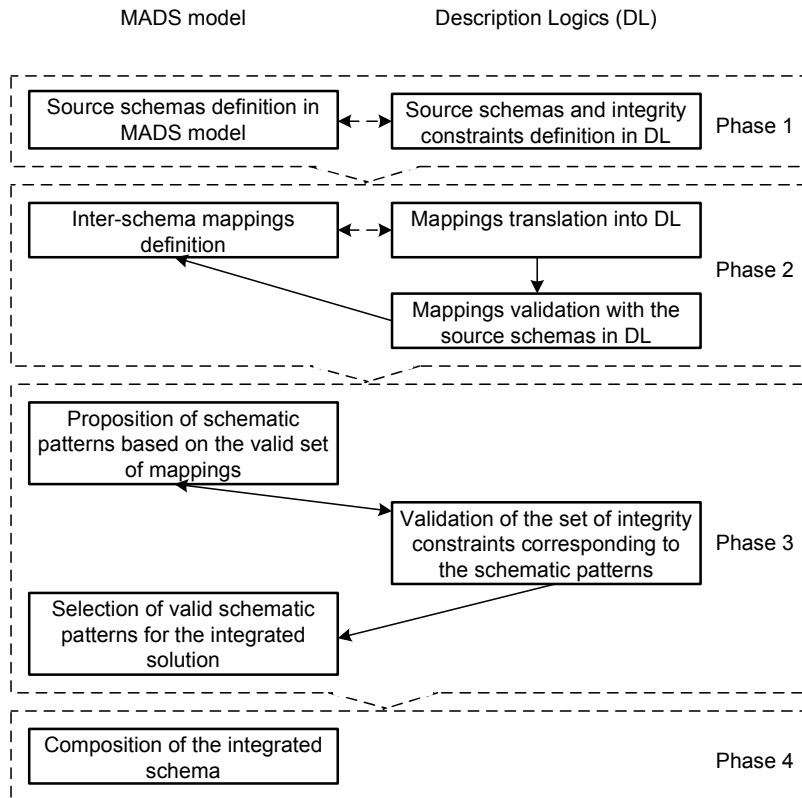


Figure 4.1: Integration phases.

and the satisfiability of the resulting translated DL model is checked. In phase 2), inter-schema mappings are added to the DL model in terms of elements of the source schemas and mapping operators [SMS02]. If the resulting model is found satisfiable, then the mappings are data model compliant and there are no contradictory mappings. Several possible ways to construct the integrated schema are then presented to the schema designer. During the phase 3) these putative structural solutions are checked for compatibility with the integrity constraints imposed on the schema elements that are to be integrated. A successful satisfiability check signifies that the populations of the related schema elements can be merged. If this check fails, only the structural solutions that model the populations separately are applicable. The final decision on the choice of the structural solutions is up to the designer and/or a domain expert. By composing our integration method of the phases described above, we ensure that the schema designer has sufficient knowledge about the compatibility of the source schemas, i.e., valid inter-schema mappings and valid structural solutions to construct a final integrated schema.

4.1 Validation in $\mathcal{ALCRP}(\mathcal{D})$

Theoretically, we are looking for the best suited description logic(s) for modeling spatial and temporal data. As presented in Section 2.4.1, there are expressive description logics that provide for modeling of spatial and temporal data. In this section, we first give a formal definition of the role-forming predicate in $\mathcal{ALCRP}(\mathcal{D})$ description logic found in [HLM99], and then describe an approach that we adopt to combine two concrete domains to model spatio-temporal data. As in MADS, for data representation in logic we aim at the orthogonal representation, where the spatial, temporal, and multi-representation dimensions are independent.

4.1.1 Description Logic $\mathcal{ALCRP}(\mathcal{D})$

The concept language $\mathcal{ALCRP}(\mathcal{D})$ is a descendant of the $\mathcal{ALC}(\mathcal{D})$ (introduced in Section 2.4.2) and it incorporates concrete domains which are relevant to our goal of modeling spatial and temporal information (the definition of the concrete domain is given earlier in Section 2.4.3). The $\mathcal{ALCRP}(\mathcal{D})$ extends the $\mathcal{ALC}(\mathcal{D})$ with the *role-forming* predicate operator to express complex topological relationships between elements of the concrete domain $\mathcal{D} = (\Delta_{\mathcal{D}}, \Phi_{\mathcal{D}})$. The definition of the *role-forming* predicate is as follows (from [HLM99]).

DEFINITION 2 *Let R and F be disjoint sets of roles and feature names where features (functional properties) relate abstract objects from $\Delta_{\mathcal{I}}$ to concrete objects from $\Delta_{\mathcal{D}}$. Any element of $R \cup F$ is an atomic role term. A composition of features (written $f_1 f_2 \dots f_n$) is called a feature chain. If $P \in \Phi_{\mathcal{D}}$ is a predicate name with arity $n + m$ and u_1, \dots, u_n as well as v_1, \dots, v_m are feature chains, then the expression $\exists(u_1, \dots, u_n)(v_1, \dots, v_m).P$ (role-forming predicate operator) is a complex role term. Let S be a role name and let T be a role term. Then $S \doteq T$ is a terminological axiom. This type of terminological axiom is also called role definition.*

The semantics of the *role-forming* predicate is as follows:

$$\begin{aligned} (\exists(u_1, \dots, u_n)(v_1, \dots, v_m).P)^{\mathcal{I}} = & \\ & \{(a, b) \in \Delta_{\mathcal{I}} \times \Delta_{\mathcal{I}} \mid \exists x_1, \dots, x_n, y_1, \dots, y_m \in \Delta_{\mathcal{D}} : \\ & (a, x_1) \in u_1^{\mathcal{I}} \wedge \dots \wedge (a, x_n) \in u_n^{\mathcal{I}} \wedge \\ & (b, y_1) \in v_1^{\mathcal{I}} \wedge \dots \wedge (b, y_m) \in v_m^{\mathcal{I}} \wedge \\ & (x_1, \dots, x_n, y_1, \dots, y_m) \in P^{\mathcal{D}}\} \end{aligned}$$

Together with the role-forming predicate semantics, in the following section we will use *terminological* axioms. In the most general case, *terminological* axioms have the form:

$$C \sqsubseteq D \quad (R \sqsubseteq S) \quad \text{or} \quad C \equiv D \quad (R \equiv S)$$

where C and D are concepts (and R, S are roles). Axioms of the first kind are called *inclusions*, while axioms of the second kind are called *equalities*. The role inclusion axiom was introduced in Section 2.4.2. The semantics of the concept axioms is defined as follows:

$$\begin{aligned} (C \sqsubseteq D)^{\mathcal{I}} &= C^{\mathcal{I}} \subseteq D^{\mathcal{I}} \\ (C \equiv D)^{\mathcal{I}} &= C^{\mathcal{I}} = D^{\mathcal{I}} \end{aligned}$$

Using the $\mathcal{ALCRP}(\mathcal{D})$ description logic we can define MADS object types as $\mathcal{ALCRP}(\mathcal{D})$ concepts, MADS attributes and relationships as $\mathcal{ALCRP}(\mathcal{D})$ roles, and use the concrete domains with concrete (complex) roles to define MADS topological and synchronization relationships.

4.1.2 Spatio-Temporal Concrete Domain

As we presented in Section 2.4.3, the concrete domains that we want to exploit for our validation phase are the spatial and temporal domains. As in MADS, we aim at modeling independently the terminological, spatial and temporal phenomena in the frame of the same formalism. Thus, we aim at exploiting a description logic with a combined, spatio-temporal domain. Contrary to the approaches where several logics are combined together, like in [KLWZ04], we want to use the $\mathcal{ALCRP}(\mathcal{D})$ with a combined domain $\mathcal{D} = \mathcal{S}_2 \oplus \mathcal{T}$. The decisive property for concrete domains to be combined, is the *admissibility* of the domains. Thus, the precondition for the combination of logics with spatial and temporal concrete domains, is the admissibility of their domains. The definition of the admissibility of a domain from [BCM⁺03]:

DEFINITION 3 *A concrete domain \mathcal{D} is admissible iif (i) the set of predicate names $\Phi_{\mathcal{D}}$ is closed under negation, i.e., for every predicate $P \in \Phi_{\mathcal{D}}$ there is a $\neg P \in \Phi_{\mathcal{D}}$ and (ii) $\Phi_{\mathcal{D}}$ contains a name $\top_{\mathcal{D}}$ for $\Delta_{\mathcal{D}}$, and (iii) the satisfiability problem $P_1^{n_1}(x_{11}, \dots, x_{1n_1}) \wedge \dots \wedge P_m^{n_m}(x_{m1}, \dots, x_{mn_m})$ is decidable (m is finite, $P_i^{n_i} \in \Phi_{\mathcal{D}}$, n_i is the arity of the P , and x_{jk} is a concrete object).*

In a description logic with a concrete domain, the concrete domain is a parameter to the DL, and as proposed in [Lut02], the sets of the abstract and concrete features are kept disjoint for a clearer algorithmic treatment. Furthermore, as it is shown in [BH91], any two admissible concrete domains \mathcal{D}_1 and \mathcal{D}_2 which have disjoint domain sets $\Delta_{\mathcal{D}_1}$ and $\Delta_{\mathcal{D}_2}$, can be combined into a new concrete domain $\mathcal{D}_1 \oplus \mathcal{D}_2$ as follows: (1) The domain $\Delta_{\mathcal{D}_1 \oplus \mathcal{D}_2}$ is the union of $\Delta_{\mathcal{D}_1}$ and $\Delta_{\mathcal{D}_2}$ and (2) the set of predicates $\Phi_{\mathcal{D}_1 \oplus \mathcal{D}_2}$ is set to $\Phi_{\mathcal{D}_1} \cup \Phi_{\mathcal{D}_2}$. In the sequel, we describe logics with spatial and temporal extensions that are used in our methodology to model spatio-temporal data.

In particular, an appropriate concrete domain \mathcal{S}_2 is defined for polygons using \mathcal{RCC}_8 relations as basic predicates of concrete domain as shown in Figure 4.2 (disjoint stands for the DC \mathcal{RCC}_8 relationship, touching for EC, s_overlapping for PO, t_inside

($t_contains$) for TPP, s_inside ($s_contains$) for NTPP, and $equal$ for EQ). In [HLM99] it is shown that the concrete domain \mathcal{S}_2 is admissible. Condition (i) of the admissibility of the domain \mathcal{S}_2 is held the set of the \mathcal{RCC}_8 relationships is closed under negation.

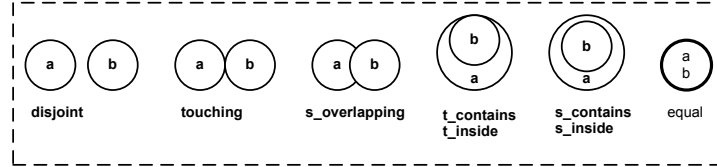


Figure 4.2: Topological relationships.

For temporal aspect, the concrete domain \mathcal{T} is a set of time intervals and the 13 Allen relationships before ($<(i,j)$), after ($>(i,j)$), meets ($m(i,j)$), met-by ($mi(i,j)$), overlaps ($o(i,j)$), overlapped-by ($oi(i,j)$), starts ($b(i,j)$), started-by ($bi(i,j)$), during ($d(i,j)$), contains ($di(i,j)$), finishes ($f(i,j)$), finished-by ($fi(i,j)$), equal ($=(i,j)$), are used as basic predicates describing the relationships between intervals. The combination of \mathcal{S}_2 and \mathcal{T} , $\mathcal{S}_2 \oplus \mathcal{T}$, defines a spatio-temporal concrete domain.

Thus, we now can exploit the $\mathcal{ALCRP}(\mathcal{S}_2 \oplus \mathcal{T})$ expressive power to describe source spatio-temporal schemas and inter-schema mappings.

4.2 MADS schemas in $\mathcal{ALCRP}(\mathcal{D})$

4.2.1 Spatio-temporal features

Using $\mathcal{ALCRP}(\mathcal{S}_2 \oplus \mathcal{T})$, we can define a concept that has a geometry with a specific concrete spatial role called **hasArea**. Further, using the **hasArea** feature, we can specify topological relationships between spatial concepts. To define a concept as a temporal concept we can use a specific concrete temporal role called **hasDuration**. Depending on the cardinality of the role **hasDuration** it models different MADS temporal types (cf. Figure 3.5), i.e., **simple time** with the cardinality 1:1, **complex time** with the cardinality 1:n. Through this role, temporal relationships between concepts can then be defined. For example, elements of the schema \mathbb{T}_1 in Figure 3.12 can be described as follows.

A city has a name, it is decomposed in districts and it runs transport means:

$$\begin{aligned} \text{City} &\sqsubseteq \forall \text{Name.String} \\ &\quad \sqcap \forall \text{run.TransportMean} ; \end{aligned}$$

A tourist place has a name, it is a spatio-temporal object thus, it has a geometry and a temporality which are respectively specified by the concrete roles **hasArea** of the domain **Polygon** and **hasDuration** of the domain **Interval**:

$$\text{TouristPlace} \sqsubseteq \forall \text{Name.String}$$

$$\begin{aligned} &\sqcap \exists \text{hasArea.Polygon} \\ &\sqcap \exists \text{hasDuration.Interval} ; \end{aligned}$$

Museums are tourist places:

$$\text{Museum} \sqsubseteq \text{TouristPlace} ;$$

Monuments are tourist places having a specific feature expressing their style, with the cardinality stating that a monument has exactly one style:

$$\begin{aligned} &\text{Monument} \sqsubseteq \text{TouristPlace} \\ &\sqcap \forall \text{Style.String} \end{aligned}$$

The object types `City`, `Museum`, `Monument`, `TouristPlace`, `District`, and `Transport-Mean` are modeled as abstract concepts; the relationships `decomposedIn`, `run`, and attributes `Name`, and `Style` are modeled as roles.

To define museums that are spatially connected to some monuments and whose opening times overlap, we first define a spatial predicate `connected` (a and b are two spatial instances) as the disjunction of elementary predicates, a spatial role `spatial_connected` based on the previously defined `connected` predicate, and a temporal role `duration_overlaps`. The role `spatial_connected` (respectively `duration_overlaps`) may be used to link couples of objects whose spatiality (respectively life-cycle) satisfy the `connected` (respectively `overlaps`) predicate. Then with these roles, we define such museums, `MuseumMonument`, as follows:

$$\begin{aligned} \text{connected}(a, b) &\doteq \text{touching}(a, b) \vee \text{s_overlapping}(a, b) \vee \text{t_contains}(a, b) \\ &\quad \vee \text{t_inside}(a, b) \vee \text{s_contains}(a, b) \vee \text{s_inside}(a, b) \vee \text{equal}(a, b) ; \\ \text{spatial_connected} &\doteq \exists(\text{hasArea})(\text{hasArea}).\text{connected} ; \\ \text{duration_overlaps} &\doteq \exists(\text{hasDuration})(\text{hasDuration}).\text{overlaps} ; \end{aligned}$$

$$\begin{aligned} \text{MuseumMonument} &\sqsubseteq \text{Museum} \\ &\sqcap \exists \text{spatial_connected.Monument} \\ &\sqcap \exists \text{duration_overlaps.Monument} ; \end{aligned}$$

These descriptions combine not only abstract and concrete objects but also the spatial and temporal concrete domains. This aspect ensures that reasoning can be achieved according to the intended semantics of spatio-temporal objects.

4.2.2 Inter-schema mappings

The inter-schema mappings are initially formulated in the MADS language, for details of the language the interested reader is referred to Section 3.4. We distinguish several types of MADS inter-schema mappings. Firstly, there are mappings that express the relationships between populations of schema elements that are intentionally related. We use terms intentionally and extensionally in the same sense

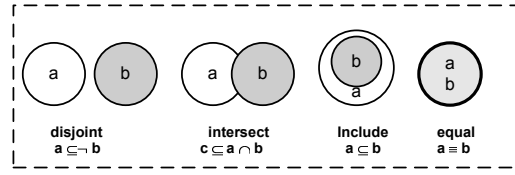


Figure 4.3: Population relationships and corresponding DL expressions.

as in [CL93], i.e., intentionally related object types share the schema level representation; extensionally related object types share parts of their populations. We call these mappings *Schema population Correspondences* or SCs. For the population correspondences we apply the set operators shown in Figure 4.3. Intuitively, if an SC is asserted, then an intentional correspondence is assumed, and the extensional correspondence is defined by the SC operator. If the operator is **disjoint**, then there are no common instances in the two populations, thus the set of the inter-schema mappings only describes the intentional correspondences.

Further, the set of Property semantic Correspondences (PCs) details correspondences between the descriptions of the schema elements involved in an SC. By the description we assume the set of attributes, including identifiers. Again, depending on the SC operator, the set of the PCs describes either the intentional correspondences (case of the **disjoint** operator), or both intentional and extensional correspondences. As the attributes and relationships in MADS can have spatial/topological and(or) temporal/synchronization semantics, the set of PC operators includes spatial operators (Figure 4.2) and a subset of Allen operators detailed in Section 2.4.1. PCs that involve identifier attributes are called *Matching Rules* or MRs. In case of a non-disjoint operator in the SC, the MRs are used to match identical instances. This set of mappings, formulated for all intentionally and extensionally related schema elements is then used for defining possible schematic patterns for an integrated schema.

Schema population Correspondences.

Hereafter we use the two schemas, T_1 and T_2 from Figures 3.12 and 3.13 respectively to illustrate the integration process. Analyzing the description of the two schemas that is found in Section 3.3, the schema designer deduces that the populations of the object types $TouristPlace_{T_1}$ and $TouristSite_{T_2}$ are not disjoint as we assume some museums and monuments are represented in both databases. The type of the relationship between these object types is intersection because the subtypes of $TouristSite_{T_2}$, *Theater* and *Walk*, are not modeled in T_1 . And, there is a subtype of $TouristPlace_{T_1}$ for which there is no corresponding subtype in $TouristSite_{T_2}$, i.e., the *Curiosity* subtype. The DL expression stating the SC as an the intersection of the populations of $TouristPlace_{T_1}$ and $TouristSite_{T_2}$ is $TouristPlace_{T_1} \sqcap TouristSite_{T_2}$. The populations of $Museum_{T_1}$ and $Museum_{T_2}$, $Monument_{T_1}$ and $Monument_{T_2}$ are included in each other, i.e., in description logics - $Museum_{T_2} \sqsubseteq Museum_{T_1}$, and $Monument_{T_2} \sqsubseteq Monument_{T_1}$.

Property semantic Correspondences and Matching Rules.

With the *Property semantic Correspondences* (PCs), the schema designer states the relationships between different representations (or part of representations) of the intentionally or extensionally same object and relationship types. Thus, these correspondences are formulated for all the types of the *Schema population Correspondences* (SCs) including disjoint ones. The alphabet of the language for the PCs consists of the attribute names of the schema elements involved in the SCs, in other words, the PCs unfold the SCs expressions. We have presented in detail the syntax and semantics of the property semantic correspondences and matching rules in Section 3.4.

Here we adopt the notation introduced in [HLM99], thus the temporality (life-cycle) of an object is translated into DL by the `hasDuration` property and, the spatiality by using the `hasArea` property. We assume that museums in T_1 have the role `hasDuration` that models their temporality; in T_2 , the object type `Museum` has a temporal attribute `openTime` thus, in T_2 , the museums have a role `openTime` whose domain is a temporal domain. Let us assume that these two different modeling solutions are meant to represent the same fact: a museum opening time. With such an assumption, the solution in the schema T_1 is less appropriate because the life-cycle of a museum does not bear the same semantics as the opening time for the same museum. To express the constraint that says that `openTimeT2` of `MuseumT2` is temporally equal to the temporality of `MuseumT1` we first have to define two roles based on temporal predicates as in [HLM99] :

$$\begin{aligned} \text{museum_equal}_1 &\doteq \exists(\text{openTime}_{T_2})(\text{hasDuration}).\text{equal} ; \\ \text{museum_equal}_2 &\doteq \exists(\text{hasDuration})(\text{openTime}_{T_2}).\text{equal} ; \end{aligned}$$

Then the constraint is defined as:

$$\begin{aligned} (\text{Museum}_{T_2} \sqcap \text{Museum}_{T_1}) &\sqsubseteq (\exists \text{museum_equal}_2.\text{Museum}_{T_2} \sqcap \\ &\exists \text{museum_equal}_1.\text{Museum}_{T_1}) ; \end{aligned}$$

To express spatial equality of `TouristPlaceT1` and `TouristSiteT2` - both object types have spatial extensions, we state the following expression in DL:

$$\begin{aligned} \text{area_equal} &\doteq \exists(\text{hasArea})(\text{hasArea}).\text{equal} ; \\ (\text{TouristPlace}_{T_1} \sqcap \text{TouristSite}_{T_2}) &\sqsubseteq \\ &(\exists \text{area_equal}.\text{TouristPlace}_{T_1} \sqcap \exists \text{area_equal}.\text{TouristSite}_{T_2}) ; \end{aligned}$$

The rest of the PCs for attributes of `TouristPlaceT1` and `TouristSiteT1` are:

$$\begin{aligned} \text{monument_equal}_1 &\doteq \exists(\text{hasDuration})(\text{construct}_{T_2}).\text{equal} ; \\ \text{monument_equal}_2 &\doteq \exists(\text{construct}_{T_2})(\text{hasDuration}).\text{equal} ; \\ \text{Monument}_{T_2} \sqcap \text{Monument}_{T_1} &\sqsubseteq \exists \text{monument_equal}_2.\text{Monument}_{T_2} \sqcap \\ &\exists \text{monument_equal}_1.\text{Monument}_{T_1} ; \\ \forall \text{district}_{T_2}.\text{TouristSite}_{T_2} &\sqsubseteq \forall \text{name}_{T_1}.\text{CityBorough}_{T_1} ; \\ \forall \text{name}_{T_2}.\text{TouristSite}_{T_2} &\sqsubseteq \forall \text{name}_{T_1}.\text{TouristPlace}_{T_1} ; \end{aligned}$$

The set of the PCs is complete if for all the SCs stated for the source schemas, all the pairs of elements (attributes) of object and relationship types involved, are examined for the existence of a PC between them. In Section 3.4, we have proposed the completeness check algorithm, which now can be performed by the DL reasoning service. To complete the set of PCs that are initially proposed by the integrated schema designer, an additional set based on the source schema descriptions and the set of the inter-schema mappings, is inferred by the reasoner. Completeness of reasoning means in this context that no valid deduction is left out by the inference engine. In the complete set of the PCs the designer can now state a subset called *Matching Rules* (MRs). The MRs are the rules that state the correspondences between instances that are represented differently in source schemas. These rules involve identifier attributes. Matching rules are useful in order to find corresponding instances during the data integration process. For our example, the MRs between $\text{TouristPlace}_{T_1}$ and TouristSite_{T_2} are those involving Name_{T_2} , Name_{T_1} attributes and the spatiality of the object types.

$$\begin{aligned} \text{TouristPlace}_{T_1} \sqcap \text{TouristSite}_{T_2} &\sqsubseteq \\ &\exists \text{area_equal}.\text{TouristPlace}_{T_1} \sqcap \exists \text{area_equal}.\text{TouristPlace}_{T_2} ; \\ \forall \text{Name}_{T_2}.\text{TouristSite}_{T_2} &\sqsubseteq \forall \text{Name}_{T_1}.\text{TouristPlace} ; \end{aligned}$$

Validation in DL.

As it was mentioned above, we use DL reasoning services to check the satisfiability of our DL model, i.e., the compatibility of the two source schemas, and the set of inter-schema mappings expressed in DL. If our model is found to be unsatisfiable, then the set of the inter-schema mappings should be reconsidered for unsatisfied objects (*Phase 2* in Figure 4.1). Unsatisfiability means that there are some concepts that describe an empty set of instances. For our example an unsatisfiable model would be detected for the following set of definitions:

Stops in Stop_{T_1} are spatially connected to bus lines in BusLine_{T_1} 's:

$$\begin{aligned} \text{stopServes}_{T_1} &\doteq \exists(\text{hasArea})(\text{hasArea}).\text{s_contains} ; \\ \text{Stop}_{T_1} &\sqsubseteq \exists \text{stopServes}_{T_1}.\text{BusLine}_{T_1} ; \end{aligned}$$

Stops in Stop_{T_2} are spatially connected to transport lines in $\text{TransportLine}_{T_2}$'s:

$$\begin{aligned} \text{along}_{T_2} &\doteq \exists(\text{hasArea})(\text{hasArea}).\text{s_contains} ; \\ \text{Stop}_{T_2} &\sqsubseteq \exists \text{along}_{T_2}.\text{TransportLine}_{T_2} ; \end{aligned}$$

Some stops are represented in both databases described by T_1 and T_2 :

$$\text{Stop}_{T_1} \sqsubseteq \text{Stop}_{T_2} ;$$

There is no transport line in $\text{TransportLine}_{T_2}$ that is spatially connected to a bus

line in BusLine_{T_1} :

$\text{area_disjoint} \doteq \exists(\text{hasArea})(\text{hasArea}).\text{disjoint}$;
 $\text{TransportLine}_{T_2} \sqsubseteq \exists \text{area_disjoint}.\text{BusLine}_{T_1}$;

This model is invalidated by the reasoner based on the following inferences: firstly, since **BusLine** and **Stop** from schema T_1 are spatially connected then, **TransportLine** and **Stop** from schema T_2 are also spatially connected. Furthermore, some **Stops** from T_1 and T_2 are the same, and consequently, some of the **BusLine** $_{T_1}$ are spatially connected to **TransportLine** $_{T_2}$. This last deduction of the reasoner contradicts the last expression of the model above.

Upon completion of this phase, the schema designer has in hand a complete and valid set of inter-schema mappings. We are now able to define a set of possible structural solutions for the integrated schema from the *Schema population Correspondences*. In the next phase (*Phase 3* in Figure 4.1), different schematic patterns are validated against the compatibility of integrity constraints for the integrated solutions.

4.2.3 Structural solution for the integrated schema

As we have presented in Section 3.4.2, proposed schematic patterns for the integrated schema suggest application of a particular structural transformation of the schema elements involved in the inter-schema mappings (cf. Figure 3.17 in Section 3.4.2). These structural transformations should be validated for the integrity of the resulting schema. The question to be answered is, whether these transformations would lead to a violation of the integrity constraints imposed on one or several schemas' elements. If the planned structural transformation is not valid for the given integrity constraints, then the integrity constraints are weakened, or another structural solution is proposed, and the check is run again. To ensure the meaningful integrated solution even for the cases of greatly diverse representations of related data we employ the multi-perception solution consistently preserving the initial representations on the integrated level.

In the following sections we consider the integration of two object types, **TouristPlace** $_{T_1}$, and **TouristSite** $_{T_2}$. We assume the following set of correspondences is stated (as explained in Section 4.2.2):

$\text{area_equal} \doteq \exists(\text{hasArea})(\text{hasArea}).\text{equal}$;
 $\text{museum_equal}_1 \doteq \exists(\text{openTime}_{T_2})(\text{hasDuration}).\text{equal}$;
 $\text{museum_equal}_2 \doteq \exists(\text{hasDuration})(\text{openTime}_{T_2}).\text{equal}$;
 $\text{monument_equal}_1 \doteq \exists(\text{hasDuration})(\text{construct}_{T_2}).\text{equal}$;
 $\text{monument_equal}_2 \doteq \exists(\text{construct}_{T_2})(\text{hasDuration}).\text{equal}$;

Schema population correspondences:

- (1) $\text{SharedTouristSite} \sqsubseteq \text{TouristPlace}_{T_1} \sqcap \text{TouristSite}_{T_2}$;
- (2) $\text{Museum}_{T_2} \sqsubseteq \text{Museum}_{T_1}$;

$$(3) \text{ Monument}_{T_2} \sqsubseteq \text{Monument}_{T_1} ;$$

Property semantic correspondences:

$$(4) \text{Museum}_{T_2} \sqcap \text{Museum}_{T_1} \sqsubseteq \exists \text{museum_equal}_2. \text{Museum}_{T_2} \sqcap \exists \text{museum_equal}_1. \text{Museum}_{T_1} ;$$

$$(5) \text{Monument}_{T_2} \sqcap \text{Monument}_{T_1} \sqsubseteq \exists \text{monument_equal}_2. \text{Monument}_{T_2} \sqcap \exists \text{monument_equal}_1. \text{Monument}_{T_1} ;$$

$$(6) \forall \text{district}_{T_2}. \text{TouristSite}_{T_2} \sqsubseteq \forall \text{name}_{T_1}. \text{CityBorough}_{T_1} ;$$

Matching Rules:

$$(7) \text{TouristPlace}_{T_1} \sqcap \text{TouristSite}_{T_2} \sqsubseteq \exists \text{area_equal}. \text{TouristPlace}_{T_1} \sqcap \exists \text{area_equal}. \text{TouristSite}_{T_2} ;$$

$$(8) \forall \text{name}_{T_2}. \text{TouristSite}_{T_2} \sqsubseteq \forall \text{name}_{T_1}. \text{TouristPlace}_{T_1} ;$$

Schematic Patterns.

The set of possible schematic patterns depends on the type of the SCs between the related representations. As we described in Section 3.4, from the spectrum of the structural patterns [Dup94], the integrated schema designer is provided with several patterns for validation. These patterns are applied depending on the policy chosen by the designer (cf. Section 3.4.2): **fusion** - the one resulting in the lowest number of schema elements for the integrated schema; **generalization-partition** - the one that produces the most detailed integrated schema; and two types of **multi-representations** that relate source schemas without changing their structures. For our example schemas, the population correspondence on **TouristSite** and **TouristPlace** is **intersect** (as per assertion (1)), and hence the designer is provided with all four patterns. The set of available patterns would be different for different operators in the population correspondence expression. For example, with the **disjoint** operator, the **generalization-partition** pattern is excluded as its application requires common instances in related populations.

The first solution (Figure 4.4.a) is to extract the overlapping part of the populations and model it as the subtype of the two source populations. This policy uses the multi-inheritance paradigm of the MADS data model. This pattern is called **generalization-partition**. With this structural pattern, the population of the $\text{SharedTouristSite}_{T_{\text{int}}}$ is $\text{TouristPlace}_{T_1} \sqcap \text{TouristSite}_{T_2}$. The population of the $\text{SharedTouristSite}_{T_{\text{int}}}$ are those tourist sites (only of subtypes **Museum** and **Monument**) that are close to a public transport stop, i.e., accessible by the public transport.

According to the schema population correspondences (1), (2) and (3), we have an integrated representation for common entities **SharedMonument** and **SharedMuseum** for schemas T_1 and T_2 . The subtype **Curiosity** (as well as **Theatre** and **Walk**) is not present as a subtype of $\text{SharedTouristSite}_{T_{\text{int}}}$ because there are no entities of this type neither in $\text{Curiosity}_{T_1} \sqcap \text{TouristSite}_{T_2}$ nor in the $\text{TouristSite}_{T_2} \sqcap \neg \text{Curiosity}_{T_1}$.

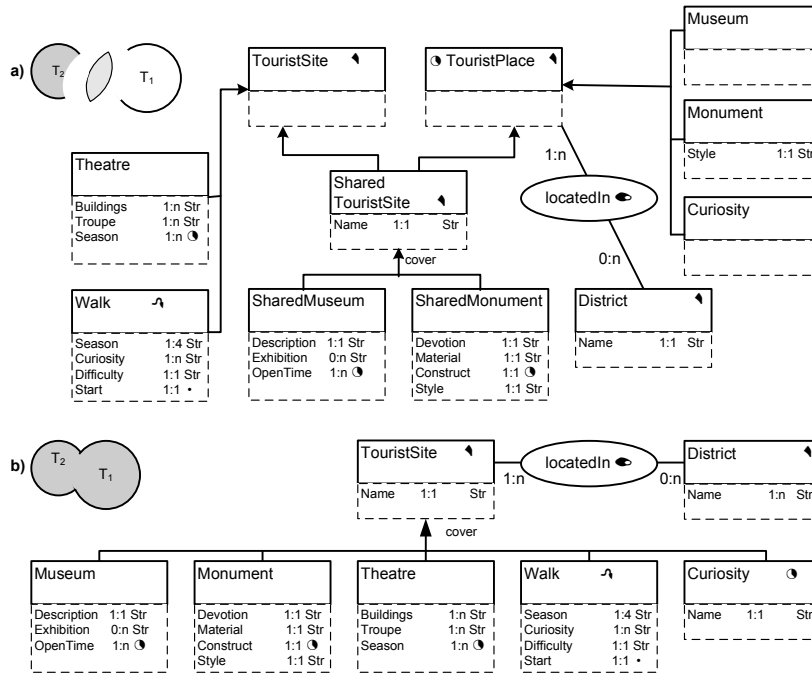


Figure 4.4: Schematic solutions under the intersection relation between the populations of the source schemas for integrated schema T_{int} .

From the correspondence assertion stating that the name of the city borough is equal to the district of the **TouristSite** (assertion (6)), the designer should decide whether he chooses to keep the modeling solution of T_1 - with an object type **CityBorough** or **District**, or the modeling solution of T_2 - with an attribute **district**. In Figure 4.4.a, city boroughs are modeled by a spatial object type **District** with **Name** attribute. The cardinality of the **locatedIn** relationship is preserved as it is in schema T_1 - a tourist site can be located in several city boroughs. Such a cardinality would be required for example for the Opera de Paris theatre, that has two buildings, one is in the 9^{eme} city borough, and another in the 12^{eme}. In schema T_2 the cardinality of the **District** attribute was 1:1, but preserving this cardinality would invalidate the extension (population) of T_1 . Another solution for **CityBorough** would be to keep it as a multi-valued attribute of **TouristSite** (as it is in T_2), but since in T_1 there are relationships attached to the **CityBorough** object type, the designer should adhere to the pattern shown in Figure 4.4.a. where the relationship **locatedIn** is linked to **TouristSite** (as in the source schema T_1 for the object type **TouristPlace**) and attribute **District** is removed from **TouristSite**. Generally, if there are several structural representations for the same real world phenomena, the designer should adhere to a less restricted one. In the Table 4.1 we describe the restrictions over different structural elements used in the MADS model as well as in any ER-based model.

Finally, we have to consider the correspondence assertions (4) and (5) about the temporality of **TouristPlace** and the temporal attributes of **Museum** and **Monument**.

Table 4.1: Conceptual model element description

Element	Restrictions
Object type	can have unlimited number of the relationship types associated to it as well as any number of attributes.
Relationship type	must have two object types associated to it; can have any number of attributes.
Attribute	must have an object type as its container.

To be consistent with the schema T_1 , the temporality of **TouristPlace** should be preserved. Considering the MADS model, several solutions are possible for the temporal attributes **openTime** and **construct**: we could either remove them as the temporality of **TouristPlace** is inherited in **Museum** and **Monument**, or define them as derived attributes (derived from the inherited temporality) or finally, keep them and add an integrity constraint. In Figure 4.4.a, we choose to present the second possibility with derived attributes to keep the resulting schema more detailed.

The second possible structural pattern (Figure 4.4.b) is **fusion** where the populations of the source schemas are merged. As previously, before giving the final schema, the designer has to consider the correspondence assertions (6), (4) and (5). The proposed solution for the **CityBorough** is the same as in the first pattern for the same reasons. Considering the temporality of the **TouristSite** object type, the situation and the proposed pattern are different from above: the temporality of **TouristPlace** $_{T_1}$ is migrated one level down (**TouristPlace** no longer has a temporality but all its subtypes have one), because in T_2 there are more subtypes for **TouristSite** and not all of them have temporal attributes. In addition, usage of the redefined temporal attributes **OpenTime** $_{T_{int}}$ and **Construct** $_{T_{int}}$ is more expressive for the schema user than would be the inherited temporality (as it was in the source schema T_1).

Finally, the third and fourth possible structural solutions are the **multi-representations** shown in Figure 4.5, where the initial representations and local integrity constraints are preserved and no structural transformation is done. This pattern can be applied in the situation where all other proposed patterns are invalidated.

Two possible modeling solutions may be considered: the designer could either choose to link the different object types under consideration with a link holding the specific inter-representation semantics (as in Figure 4.5.c) or integrate the different representations in a multi-representation object type (as in Figure 4.5.d). The last solution is structurally the same as the fusion but all the schema elements hold the stamps characterizing the schema from where they come: **t1** for elements described in T_1 , and **t2** for elements from T_2 . Thus, the object type **TouristSite** holds the stamps **t1**, **t2** as it is defined in both schemas (with a different name but the same semantic) whereas **Curiosity** bears only the stamp **t1** as it is only described in T_1 . When considering the object **Monument** stamped **t1**, **t2**, its attributes **Devotion** and

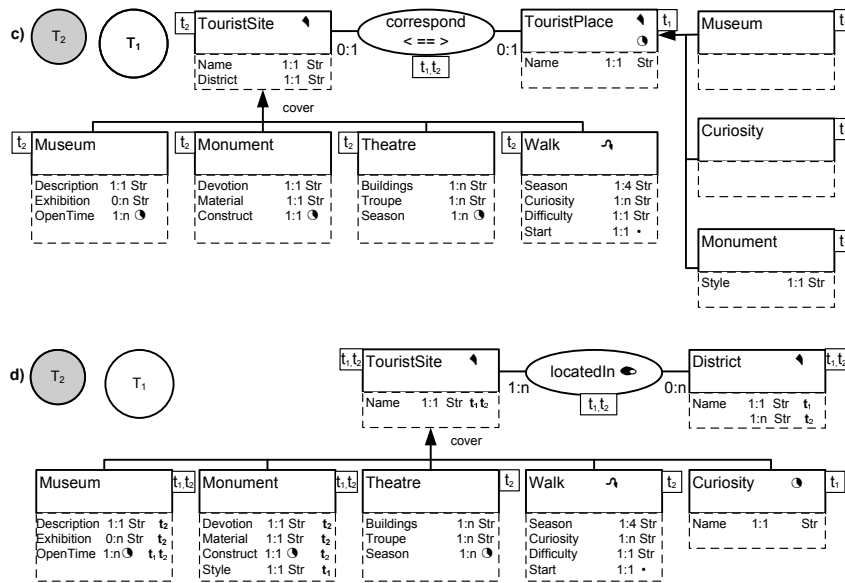


Figure 4.5: Multi-representation solutions under the intersection relation between the populations of the source schemas for integrated schema T_{int} .

Material are stamped t_2 as these attributes are only described in the schema T_2 , Style is stamped t_1 and finally Construct is defined in both schemas thus stamped t_1, t_2 . Moreover, the object District is stamped t_1, t_2 and its attribute Name has a representation-varying definition: for t_1 it is a mono-valued attribute and for t_2 it is a multi-valued attribute.

4.2.4 Composing the Integrated Schema

After the completion of the validation procedure for the DL integrated schema descriptions, the designer of the integrated schema has valid structural solutions for the related representations assured by the reasoning engine; associated integrity constraints; and mappings for all related elements of the schemas provided by the complete set of inter-schema mappings. In this last phase of the integration process the designer can choose the integrated solutions for each related element of the source schemas and compose the resulting integrated schema.

As it was shown in Section 4.2.3, for each set of mappings, a designer is provided with one or more valid structural solutions (Figures 4.4 and 4.5 show possible structural solutions for **TouristPlace** and **TouristSite** object types). For the final integrated schema, for each set of mappings for (at least) intentionally related object types, the schema designer can choose one of the solutions following a criterion. This criterion is application dependent and could be for example, the complexity of the structural solution, or the type of links used, or the types of queries that will be processed by the information system under development. Considering the structural solutions

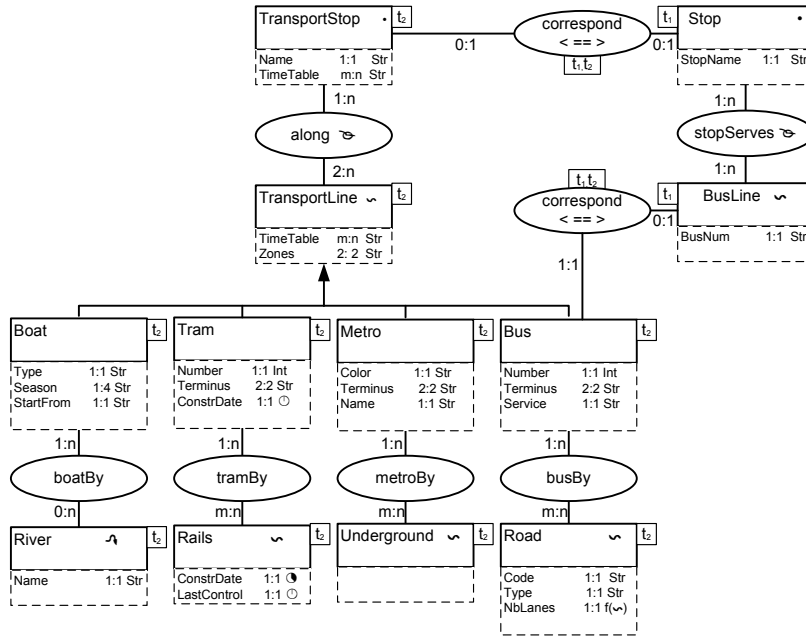


Figure 4.6: Multi-representation solution for bus lines and bus stops in an T_{int} .

shown in Figures 4.4 and 4.5, the designer could choose the **fusion** as the least complex one, i.e., the one with the lowest number of elements. As the solution for the bus lines and bus stops representation, the designer could choose to adhere to the **multi-representation** solution as shown in Figure 4.6, to avoid the invalidation of the population of BusLine_{T_1} object type (cf. the validation section above).

Let us now illustrate how the mappings can be used in an integrated database. For the part of the schema shown in Figure 4.6 the following mappings for the **Bus** and BusLine_{T_1} object types are relevant:

Schema population correspondences:

(1) $\text{Bus}_{T_2} \sqsubseteq \text{BusLine}_{T_1}$;

Matching Rules within the Property semantic correspondences:

(2) $\forall \text{number}.\text{Bus}_{T_2} \sqsubseteq \forall \text{busNum}.\text{BusLine}_{T_1}$;

Let us assume that we use an SQL like query language to query and maintain the database. Mapping (1) requires insertions of an equal instance in the **BusLine** table every time a new instance is inserted in the **Bus** table. The equality between the instances of **Bus** and **BusLine** is defined by the mapping (2), i.e., the value of the attribute **busNum** in **BusLine** must be equal to the value of the attribute **number** in **Bus**. Then, the following code will be added to keep the integrated database valid:

```
CREATE TRIGGER busLine
AFTER INSERT ON Bus BEGIN
    SELECT busNum FROM BusLine WHERE busNum = NEW.number
    IF SQL%NOTFOUND THEN INSERT INTO BusLine VALUES(NEW.number) ;
    ENDIF ;
END ;
```

Now, every time an insert operation is executed on the `Bus` table, the trigger will be fired to check the existence of an equal instance in the table `BusLine`, if it is not found, an insert operation will be executed on the `BusLine` table, with the value of the `busNum` attribute equal to the last inserted value of the `number` attribute.

4.3 Chapter Summary

We have described our approach with validation from the theoretical point of view. We have chosen the most adequate description logics to represent spatio-temporal aspects of MADS schemas. The logic we employed is the $\mathcal{ALCRP}(\mathcal{S}_2 \oplus \mathcal{T})$ with two concrete domains - \mathcal{S}_2 for MADS spatial dimension, and \mathcal{T} for MADS temporal dimension. This description logic extends the basic $\mathcal{ALC}(\mathcal{D})$ logic with role forming predicates to define complex roles over the concrete domains. The $\mathcal{ALCRP}(\mathcal{S}_2 \oplus \mathcal{T})$ features two different sets of roles - roles over abstract and roles over concrete domains. With the predefined set of concrete roles the schema designer is able to express MADS topological and synchronization relationships in $\mathcal{ALCRP}(\mathcal{S}_2 \oplus \mathcal{T})$ logic. Following the integration phases shown in Figure 4.1, after translating the data model, the schema designer states the inter-schema mappings in the same DL language.

As we mentioned earlier, one of the aims of the theoretical approach for validation was to find a formalism in which we could express the spatial and temporal aspects of the application domain. But the description logic $\mathcal{ALCRP}(\mathcal{D})$ that we have chosen lacks important constructors that are needed to represent structural aspects of databases conceptual schemas. In $\mathcal{ALCRP}(\mathcal{D})$ we could not express cardinality constraints, inverse roles, and roles hierarchies. Moreover, modeling of the spatial and temporal phenomena in $\mathcal{ALCRP}(\mathcal{D})$ is not precise enough to represent the particularities of spatial and temporal modeling with MADS. For example, in MADS, there is a hierarchy of spatial and temporal datatypes with different constraints associated to different elements of these hierarchies. Contrary, in $\mathcal{ALCRP}(\mathcal{D})$ the spatiality (temporality) is expressed using a predefined role `hasArea` (`hasDuration`) that does not convey all the semantics of spatial (temporal) dimension.

Another very important issue for our integration methodology, is the ability to use a DL reasoning service to run subsumption and satisfiability checks over the defined DL model. To the best of our knowledge, there are no reasoners implemented that are capable to capture spatial and temporal aspects of DL models. In the next chapter, we will use a more expressive description logic supported by reasoning services. The \mathcal{SHIQ} logic allows for role hierarchies, inverse and transitive roles, and qualified number restrictions but does not support spatial and temporal modeling. We will show in the next chapter how we define models in this logic to emulate spatio-temporal reasoning within MADS data model semantics.

Chapter 5

Validation: Practice

Let us now recall what are the notions or conceptions of the MADS data model and the set of inter-schema mappings that are the subject for validation. For each MADS dimension, we describe what are the rules that we want to check to provide the schema designer with a complete and correct set of the inter-schema mappings and structural patterns for the integrated schema.

Structural dimension. MADS structural dimension is based on the Extended Entity-Relationship (EER) data model. It is shown in the literature, that an EER model can be translated into the \mathcal{DLR} description logic ([CLN99],[BLR03]) and its correctness can be checked with the FaCT description logic server [FN00]. *i•com*, a conceptual modeling tool developed by the authors of [FN00], has a provably complete inference mechanism for consistency checking and for deduction, i.e., derivation of implied links in the schema. Thus, for the correctness and the completeness of the structural dimension, it is sufficient to use an available description logic reasoner. Below, we will justify our choice of the DL reasoner.

Spatial and Temporal Dimensions. Besides the structural dimension, the MADS data model implements spatial and temporal dimensions. We describe them together as they are coherently designed to provide for the orthogonal modeling approach. Considering the spatial and temporal elements of user schemas, we want to check that the spatial (temporal) elements conform to their respective type hierarchies (cf. Figures 3.3 and 3.5). A nonconforming schema element would be as shown in Figure 5.1, because in the spatial hierarchy, the type **line** is a subtype of the type **geo**. Using the terms of the object oriented theory, by imposing this hierarchical restriction, we allow for the refinement of the spatial features, but forbid their overloading. The spatial (temporal) dimensions include a predefined set of constrained relationships, i.e., topological and synchronization relationships. These relationships can be stated only between subsets of the spatial (temporal) types. For example, topological relationship **overlap** cannot be stated between two spatial types holding the **point** spatial semantic. We want to be able to check that the constrained

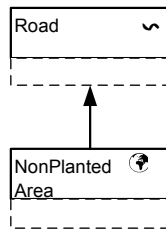


Figure 5.1: A non-conforming to the SADT hierarchy schema element.

relationships are stated only between allowed spatial (temporal) types. Some of the constrained relationships are contradictory for example, stating that a spatial object type st_1 overlaps with a spatial type st_2 , and that its subtype sub_st_1 is disjoint with the st_2 is incorrect. We want to be able to express such rules in our model.

Multi-Representation Dimension. In the methodology that we present in this thesis, we assume that the multi-representation concepts are used in the *preservation* policy, Figure 3.17, to construct the integrated schema. For the source schemas we assume that they are designed with the mono-representation premise. As all the checks are done before the construction of the integrated schema, we do not require any check on the multiple perceptions during the validation.

Inter-Schema Mappings. The set of the inter-schema mappings is stated manually by the schema designer, thus it is required to be checked for the correctness and completeness. The inter-schema mappings are expressed in the MADS correspondence language that we have presented in Section 3.4.2. Mappings in the structural dimension include set operators for related populations and the equivalence operator for related attributes. Reasoning about such types of operators falls in the category covered by the available DL reasoners. For the spatial and temporal mappings, the MADS correspondence language includes MADS set of constrained relationships as mapping operators and therefore, the same correctness rules are applied to the spatial and temporal inter-schema mappings as for the spatial and temporal MADS dimensions.

The choice of the language, the reasoner, and the editor. Considering the validation phase of our integration methodology from the practical perspective, let us start with the language that we use to express MADS schemas. There is a number of works that study the correspondences between conceptual models used for database modeling and ontologies used for data modeling for semantic web; in some, it is argued that the conceptual modeling can be done with description logics [CLN98], i.e., by designing an ontology in a description logic based language [BHS03]; other argue that the conceptual models can be used for ontology modeling [CPSV03]. To enlist the support of both camps, and provide our methodology with perspectives, we have chosen a most widely used ontology language Ontology Web Language (OWL) as

MADS counterpart. The OWL is intended to be used when the information needs to be processed by applications, as opposed to situations where the content only needs to be presented to humans. OWL can be used to explicitly represent the meaning of terms in vocabularies and the relationships between those terms. This representation of terms and their interrelationships is called an ontology. OWL has more facilities for expressing meaning and semantics than XML, RDF, and RDF-S, and thus OWL goes beyond these languages in its ability to represent machine interpretable content. OWL has three increasingly-expressive sub-languages: OWL-Lite, OWL-DL, and OWL-Full [OWL].

There are two most widely used reasoning engines, FaCT++ and RACER. FaCT++ [FaC] is an implementation of an OWL-Lite reasoner. The biggest differences between the OWL-Lite and OWL-DL abstract syntaxes is in the axioms, which are used to provide information about classes and properties. OWL-Lite supports a simpler constraints set. For example, while OWL-Lite supports cardinality constraints, it only permits cardinality values of 0 or 1. Primarily, the OWL-Lite is used to create classification hierarchies, or taxonomies. The OWL-Lite, as its name says, is the less expressive OWL sub-language.

OWL-DL, named due to its correspondence with description logics, supports the maximum expressiveness without losing computational completeness (all entailments are guaranteed to be computed) and decidability (all computations will finish in finite time) of reasoning systems. OWL-DL includes all OWL language constructs with restrictions such as type separation, e.g., a class can not be an individual or property, a property can not be an individual or class [OWL]. Reasoning about models defined in the OWL-DL sub-language, requires more advanced algorithms. Appropriate functionalities are provided by the RACER knowledge representation system [RAC]. RACER implements a highly optimized table calculus for a very expressive description logic. The system implements the *SHIQ* description logic [HST00]. This is the basic logic *ALC* augmented with qualified number restrictions, role hierarchies, inverse roles, and transitive roles. In addition to these basic features, RACER provides facilities for algebraic reasoning including concrete domains for example, min/max restrictions over integers, equalities and inequalities on strings.

Finally, for the models defined in the OWL-Full language, there is no reasoning support available. Thus, in practice we are limited by the *SHIQ* description logic and the OWL-DL sub-language. Among the available ontology design tools, e.g., OilEd, RICE, we found the knowledge-base framework Protégé [PRO] the most appropriate for our purpose of modeling and reasoning about the spatiotemporal schemas. Protégé combines a powerful ontology design tool, numerous extensions for querying, visualizing and reasoning about ontologies and it supports XML Schema, RDF(S) and OWL syntaxes. The *SHIQ* logic does not treat neither spatial, nor temporal concrete domains. In the sequel, we first show what are the correspondences between MADS and *SHIQ* constructs, and then how we have implemented MADS data model in OWL, emulating spatial and temporal rules for describing

spatio-temporal information.

5.1 MADS data model in \mathcal{SHIQ} logic

As we presented in the beginning of this chapter, we do not aim at translating complete MADS semantics into a description logic formalism. Given the expressiveness of the MADS model, it would constitute a topic for another research work. Instead, we translate the MADS concepts and rules that are necessary for the validation phase of our integration methodology. Assume we translate a MADS schema \mathcal{S} into a \mathcal{SHIQ} knowledge base \mathcal{K} . We follow the translation procedure described in [CLN99], and we adopt the definition of a \mathcal{SHIQ} knowledge base from [OCEF05]. In the sequel, we first present the definition of a \mathcal{SHIQ} knowledge base, then its semantics. Then follows the definition of a (part of) MADS schema and its semantics, and finally we present the relationship between the two models.

\mathcal{SHIQ} knowledge base.

DEFINITION 4 (\mathcal{SHIQ} knowledge base). *A \mathcal{SHIQ} knowledge base is a triple $\mathcal{K} = \langle \mathcal{A}, \mathcal{R}, \mathcal{T} \rangle$ where \mathcal{A} is an A-Box or a set of assertions, \mathcal{R} is role hierarchy and \mathcal{T} is a T-Box or terminology.*

Let \mathbf{C} be a set of concept names with subsets: $\mathbf{C}^{\text{spa}} \in \mathbf{C}$ of spatial concept names and $\mathbf{C}^{\text{tem}} \in \mathbf{C}$ temporal concept names. Let \mathbf{R} be a set of role names with subsets: $\mathbf{R}^+ \in \mathbf{R}$ of transitive role names; $\mathbf{R}^{\text{spa}} \in \mathbf{R}$ of spatial role names; and $\mathbf{R}^{\text{tem}} \in \mathbf{R}$ of temporal role names. The set of \mathcal{SHIQ} concepts is the smallest set such that:

- every concept name is a concept,
- if C and D are concepts, R is a role, S is an atomic role and n is a nonnegative integer, then $C \sqcup D$, $C \sqcap D$, $\neg C$, $\forall R.C$, $\exists R.C$, $\geq nS.C$, $\leq nS.C$ are concepts.

An assertion is an expression that can have form $C(x)$, $R(x,y)$, or $x \neq y$, where C is a concept, R is a role and $x,y \in \mathbf{I}$, where \mathbf{I} is a set of individual names.

A role hierarchy is a set of role inclusion axioms¹. The set of roles is $\mathbf{R} \cup \{R^- \mid R \in \mathbf{R}\}$. The functions Inv and Trans are defined on roles. Inv is defined as $\text{Inv}(R) = R^-$ and $\text{Inv}(R^-) = R$ for any role name R . Trans is a boolean function, $\text{Trans}(R) = \text{true}$ iff $R \in \mathbf{R}^+$ or $\text{Inv}(R) \in \mathbf{R}^+$. A role inclusion axiom is an expression of the form $R \sqsubseteq S$, where R and S are roles.

A terminology or T-Box is a set of concept inclusion axioms. A concept inclusion axiom is an expression of the form $C \sqsubseteq D$ for two concepts C and D .

¹We detail here the role hierarchy semantics introduced in Section 2.4.2.

The semantics of *SHIQ* is given by interpretations. We adopt *SHIQ* semantics definition from [TH04] and we add the value restriction for the spatial and temporal subsets of the role and concept names. The semantics is given by means of an interpretation $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$, consisting of a non empty set $\Delta^{\mathcal{I}}$, called the domain of \mathcal{I} , and a valuation $\cdot^{\mathcal{I}}$ that maps every concept to a subset of $\Delta^{\mathcal{I}}$ and every role to a subset $\Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$ such that, for all concepts C, D , roles R, S , and non-negative integer n , the semantic conditions of the logic are satisfied, where $\#M$ denotes the cardinality of the set M .

concepts and roles	syntax	semantics
atomic concept C	A	$A^{\mathcal{I}} \in \Delta^{\mathcal{I}}$
atomic role S	S	$S^{\mathcal{I}} \in \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$
inverse role	R^{-}	$\{(x, y) \mid (y, x) \in R^{\mathcal{I}}\}$
transitive role	R^{+}	$\bigcup_{i \geq 1} (R^{\mathcal{I}})^i$
role hierarchy	$R \sqsubseteq S$	$R^{\mathcal{I}} \subseteq S^{\mathcal{I}}$
concept hierarchy	$C \sqsubseteq D$	$C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$
conjunction	$C \sqcap D$	$C^{\mathcal{I}} \cap D^{\mathcal{I}}$
disjunction	$C \sqcup D$	$C^{\mathcal{I}} \cup D^{\mathcal{I}}$
negation	$\neg C$	$\Delta^{\mathcal{I}} \setminus C^{\mathcal{I}}$
exists restriction	$\exists R.C$	$\{x \mid \exists y.(x, y) \in R^{\mathcal{I}} \wedge y \in C^{\mathcal{I}}\}$
value restriction	$\forall R.C$	$\{x \mid \forall y.(x, y) \in R^{\mathcal{I}} \rightarrow y \in C^{\mathcal{I}}\}$
spatial	$\forall R^{\text{spa}}.C$	$\{x \mid \forall y.(x, y) \in R^{\text{spa}\mathcal{I}} \rightarrow y \in C^{\text{spa}\mathcal{I}}\}$
temporal	$\forall R^{\text{tem}}.C$	$\{x \mid \forall y.(x, y) \in R^{\text{tem}\mathcal{I}} \rightarrow y \in C^{\text{tem}\mathcal{I}}\}$
at least restriction	$\geq nS.C$	$\{x \mid \# \{y.(x, y) \in S^{\mathcal{I}}\} \wedge y \in C^{\mathcal{I}} \geq n\}$
at most restriction	$\leq nS.C$	$\{x \mid \# \{y.(x, y) \in S^{\mathcal{I}}\} \wedge y \in C^{\mathcal{I}} \leq n\}$

MADS schema.

In Section 3.2, we have introduced MADS data model. Hereafter, for the purpose of the translation we present a partial semantics of the model that is of interest for the validation procedures in our integration method.

From [PSZ06], a MADS schema $S \in \mathbf{S}$ is a tuple $S = (\mathbf{Name}, \mathbf{MetaData}, \mathbf{ObjectTypes}, \mathbf{RelationshipTypes}, \mathbf{MultiInstantiation})$. Where, \mathbf{S} is a set of schemas; \mathbf{Name} is the name of the schema; $\mathbf{MetaData}$ defined schema meta-data as for example its spatial extent, temporal extent, and schema perceptions; $\mathbf{ObjectTypes}$ is a set of schema object types definitions'; $\mathbf{RelationshipTypes}$ is a set of schema relationships definitions'; and $\mathbf{MultiInstantiation}$ defines to which extent object and relationships may be multi-instantiated within different object/relationship types in the schema. From the schema definition, we are interested in $\mathbf{ObjectType}$ and $\mathbf{RelationshipType}$ definitions.

An object type $O \in \mathbf{O}$ is a tuple $O = (\mathbf{Name}, \mathbf{Ostamps}, \mathbf{Supertypes}, \mathbf{Lifecy-}$

cle, Geometry, Attributes, Attributes+, Methods, Methods+, Domain, Keys). For an object type, we are interested in its **Attributes** set $A_O \in \mathbf{A}$, that may include two peculiar attributes **LifeCycle** and **Geometry**, that if present define the object type as temporal, spatial, or temporal and spatial, and the characteristics of its spatiality and temporality.

A relationship type $R \in \mathbf{R}$ is a tuple $R = (\mathbf{Name}, \mathbf{Rstamps}, \mathbf{Supertypes}, \mathbf{MultiAssociation}, \mathbf{Semantics}, \mathbf{Semantics+}, \mathbf{Roles}, \mathbf{Roles+}, \mathbf{Geometry}, \mathbf{LifeCycle}, \mathbf{Attributes}, \mathbf{Attributes+}, \mathbf{Methods}, \mathbf{Methods+}, \mathbf{Domain}, \mathbf{Keys})$. In the definition of a relationship type, similarly to the object type definition, we are interested in its **Attributes** set $A_R \in \mathbf{A}$. Additionally, the **Roles** function that defines for each perception stamp of R the local set of roles, provides us with the **ObjectCardinality** set for the relationship type R . Using the schema elements needed for validation, we give below a definition of the syntax of a MADS schema. In the following, for two finite sets X and Y , a function from a subset of X to Y is called an X -labeled tuple over Y . The labeled tuple T that maps $x_i \in X$ to $y_i \in Y$, for $i \in \{1, \dots, k\}$, is denoted as $[x_1 : y_1, \dots, x_k : y_k]$, and $T[x_i]$ denotes y_i .

DEFINITION 5 (MADS schema.) *A (partial) MADS schema is a tuple $\mathcal{S} = (\mathcal{L}_{\mathcal{S}}, \preceq_{\mathcal{S}}, \text{atts}_{\mathcal{S}}, \text{rels}_{\mathcal{S}}, \text{cards}_{\mathcal{S}})$.*

$\mathcal{L}_{\mathcal{S}}$ is a finite alphabet partitioned into a set $\mathcal{O}_{\mathcal{S}}$ of pairwise disjoint object type symbols, a set $\mathcal{A}_{\mathcal{S}}$ of pairwise disjoint attributes symbols, a set $\mathcal{U}_{\mathcal{S}}$ of pairwise disjoint role symbols, a set $\mathcal{R}_{\mathcal{S}}$ of pairwise disjoint relationship symbols, and a set $\mathcal{D}_{\mathcal{S}}$ of pairwise disjoint domain symbols. Each domain symbol \mathcal{D} has an associated predefined basic domain $\mathcal{D}^{\mathcal{B}\mathcal{D}}$, and these basic domains are disjoint. $\mathcal{O}_{\mathcal{S}}$ is further partitioned into: a set $\mathcal{O}_{\mathcal{S}}^{\text{spa}} \in \mathcal{O}_{\mathcal{S}}$ of spatial object types; a set $\mathcal{O}_{\mathcal{S}}^{\text{tem}} \in \mathcal{O}_{\mathcal{S}}$ of temporal object types. The similar partition applies to $\mathcal{R}_{\mathcal{S}}, \mathcal{A}_{\mathcal{S}}, \mathcal{U}_{\mathcal{S}}$, and $\mathcal{D}_{\mathcal{S}}$ subsets of $\mathcal{L}_{\mathcal{S}}$.

$\preceq_{\mathcal{S}} \subseteq \mathcal{O}_{\mathcal{S}} \times \mathcal{O}_{\mathcal{S}}$ is a binary relation over $\mathcal{O}_{\mathcal{S}}$.

$\text{atts}_{\mathcal{S}}$ is a function that maps each object type symbol in $\mathcal{O}_{\mathcal{S}}$ to an $\mathcal{A}_{\mathcal{S}}$ -labeled tuple over $\mathcal{D}_{\mathcal{S}}$.

$\text{rels}_{\mathcal{S}}$ is a function that maps each relationship symbol in $\mathcal{R}_{\mathcal{S}}$ to an $\mathcal{U}_{\mathcal{S}}$ -labeled tuple over $\mathcal{O}_{\mathcal{S}}$.

$\text{cards}_{\mathcal{S}}$ is a function from $\mathcal{O}_{\mathcal{S}} \times \mathcal{R}_{\mathcal{S}} \times \mathcal{U}_{\mathcal{S}}$ to $\mathbb{N}_0 \times (\mathbb{N}_0 \cup \{\infty\})$ that satisfies the following condition: for a relationship $R \in \mathcal{R}_{\mathcal{S}}$ such that $\text{rels}_{\mathcal{S}}(R) = [U_1 : O_1, \dots, U_k : O_k]$, $\text{cards}_{\mathcal{S}}(O, R, U)$ is defined only if $U = U_i$ for some $i \in \{1, \dots, k\}$, and if $O \preceq_{\mathcal{S}}^* O_i$, where $\preceq_{\mathcal{S}}^*$ denotes the reflexive transitive closure of \preceq . The first component of $\text{cards}_{\mathcal{S}}(O, R, U)$ is denoted as $\text{cmin}_{\mathcal{S}}(O, U, R)$ and the second component as $\text{cmax}_{\mathcal{S}}(O, U, R)$. Default values for $\text{cmin}_{\mathcal{S}}$ and $\text{cmax}_{\mathcal{S}}$ are 0 and ∞ respectively.

Intuitively, in the definition above the relation $\preceq_{\mathcal{S}}$ models the *IsA*-relationship

between object types; the function att_S is used to model *attributes* of an object type; the function rel_S associates a set of *roles* to each relationship symbol R , determining implicitly also the *arity* of R ; the function $card_S$ specifies *cardinality constraints*.

The semantics of a MADS schema is given by specifying legal database states for S . A legal database state of schema S is the set of instances of each object type and relationship type of S that satisfies all the constraints defined by S . Formally, a database state \mathcal{B} corresponding to a (partial) MADS schema $\mathcal{S} = (\mathcal{L}_S, \preceq_S, att_S, rel_S, card_S)$ is constituted by a non-empty finite set $\Delta^{\mathcal{B}}$, disjoint from all basic domains, and a function $\cdot^{\mathcal{B}}$ that maps:

- every domain symbol $D \in \mathcal{D}_S$ to the corresponding basic domain $\mathcal{D}^{\mathcal{B}_D}$,
- every object type $O \in \mathcal{O}_S$ to a subset $\mathcal{O}^{\mathcal{B}}$ of $\Delta^{\mathcal{B}}$,
- every attribute $A \in \mathcal{A}_S$ to a set $A^{\mathcal{B}} \subseteq \Delta^{\mathcal{B}} \times \bigcup_{D \in \mathcal{D}_S} \mathcal{D}^{\mathcal{B}_D}$, and
- every relationship $R \in \mathcal{R}_S$ to a set $R^{\mathcal{B}}$ of U_S -labeled tuples over $\Delta^{\mathcal{B}}$.

The elements of $\mathcal{O}^{\mathcal{B}}$, $A^{\mathcal{B}}$, $R^{\mathcal{B}}$ are called *instances* of O , A , and R respectively.

DEFINITION 6 A database state \mathcal{B} is said to be **legal** for a (partial) MADS schema $\mathcal{S} = (\mathcal{L}_S, \preceq_S, att_S, rel_S, card_S)$, if it satisfies the following conditions:

- for each pair of object types $O_1, O_2 \in \mathcal{O}_S$ such that $O_1 \preceq_S O_2$, it holds that $\mathcal{O}_1^{\mathcal{B}} \subseteq \mathcal{O}_2^{\mathcal{B}}$.
- for each pair of relationships $R_1, R_2 \in \mathcal{R}_S$ such that $R_1 \preceq_S R_2$, it holds that $R_1^{\mathcal{B}} \subseteq R_2^{\mathcal{B}}$.
- for each object type $O \in \mathcal{O}_S$, if O has an attribute $A \in \mathcal{A}_S$ with domain $D \in \mathcal{D}_S$, then for each instance $o \in \mathcal{O}^{\mathcal{B}}$ there is exactly one element $a \in A^{\mathcal{B}}$ with o as the first component, and the second component of a is an element of $\mathcal{D}^{\mathcal{B}_D}$.
- for each relationship $R \in \mathcal{R}_S$ of arity n between object types O_1, \dots, O_n , to which R is connected by means of roles U_1, \dots, U_n respectively, all instances of R are of the form $[U_1 : o_1, \dots, U_n : o_n]$, where $o_i \in \mathcal{O}_i^{\mathcal{B}}$, $i \in \{1, \dots, n\}$.
- for each relationship $R \in \mathcal{R}_S^{\text{spa}}$ between object types O_1, \dots, O_n , to which R is connected by means of roles U_1, \dots, U_n respectively, all instances of R are of the form $[U_1 : o_1, \dots, U_n : o_n]$, where $U_i \in \mathcal{U}_S^{\text{spa}}$ and $o_i \in \mathcal{O}_i^{\mathcal{B}} \wedge \mathcal{O}_i^{\mathcal{B}} \in \mathcal{O}_S^{\text{spa}}$ and $i \in \{1, \dots, n\}$.
- for each relationship $R \in \mathcal{R}_S^{\text{tem}}$ between object types O_1, \dots, O_n , to which R is connected by means of roles U_1, \dots, U_n respectively, all instances of R are of the form $[U_1 : o_1, \dots, U_n : o_n]$, where $U_i \in \mathcal{U}_S^{\text{tem}}$ and $o_i \in \mathcal{O}_i^{\mathcal{B}} \wedge \mathcal{O}_i^{\mathcal{B}} \in \mathcal{O}_S^{\text{tem}}$ and $i \in \{1, \dots, n\}$.

- for each role $U \in \mathcal{U}_S$ of relationship $R \in \mathcal{R}_S$ associated to object type $O \in \mathcal{O}_S$, and for each instance $o \in \mathcal{O}^B$ it holds that
$$cmin_S(O, U, R) \leq \#\{r \in R^B \mid r[U] = o\} \leq cmax_S(O, U, R).$$

Relationship between MADS and \mathcal{SHIQ} . We now show how the semantics of a (partial) MADS schema can be captured in \mathcal{SHIQ} by defining a translation ϕ from MADS schemas to \mathcal{SHIQ} knowledge bases, and then establishing a correspondence between legal database states and models of the derived knowledge base.

DEFINITION 7 Let $\mathcal{S} = (\mathcal{L}_S, \preceq_S, att_S, rel_S, card_S)$ be a (partial) MADS schema. The \mathcal{SHIQ} knowledge base \mathcal{K} is defined by the function $\phi(\mathcal{S}) = (\mathcal{A}, \mathcal{R}, \mathcal{T})$ as follows. The set \mathcal{A} of atomic concepts of $\phi(\mathcal{S})$ contains the following elements:

- for each domain symbol $D \in \mathcal{D}_S$, an atomic concept $\phi(D)$;
- for each object type $O \in \mathcal{O}_S$, an atomic concept $\phi(O)$;
- for each relationship type $R \in \mathcal{R}_S$, an atomic concept $\phi(R)$.

The set \mathcal{R} of atomic roles of $\phi(\mathcal{S})$ contains the following elements:

- for each attribute $A \in \mathcal{A}_S$, an atomic role $\phi(A)$;
- for each relationship $R \in \mathcal{R}_S$ such that $rel_S = [U_1 : O_1, \dots, U_k : O_k]$, k atomic roles $\phi(U_1), \dots, \phi(U_k)$.

The set \mathcal{T} of assertions of $\phi(\mathcal{S})$ contains the following elements:

- for each pair of object types $O_1, O_2 \in \mathcal{O}_S$ such that $O_1 \preceq_S O_2$, the assertion $\phi(O_1) \preceq \phi(O_2)$;
- for each object type $O \in \mathcal{O}_S$ such that $att_S(O) = [A_1 : D_1, \dots, A_k : D_k]$, the assertion
$$\phi O \preceq \forall \phi(A_1). \phi(D_1) \sqcap \dots \sqcap \phi(A_k). \phi(D_k) \sqcap \exists^=1 \phi(A_1) \sqcap \dots \sqcap \exists^=1 \phi(A_k)^2$$
;
- for each relationship $R \in \mathcal{R}_S$ such that $rel_S(R) = [U_1 : O_1, \dots, U_k : O_k]$ for $i \in \{1, \dots, k\}$, and for each object type $O \in \mathcal{O}_S$ such that $O \preceq_S^* O_i$,
 - if $m = cmin_S(O, R, U_i) \neq 0$, the assertion $\phi(O) \preceq \exists^{\geq m} (\phi(U_i))^-$;
 - if $n = cmax_S(O, R, U_i) \neq \infty$, the assertion $\phi(O) \preceq \exists^{\leq n} (\phi(U_i))^-$;

For each pair of symbols $X_1, X_2 \in \mathcal{O}_S \cup \mathcal{R}_S \cup \mathcal{D}_S$ such that $X_1 \neq X_2$ and $X_1 \in \mathcal{R}_S \cup \mathcal{D}_S$, the assertion: $\phi(X_1) \preceq \neg \phi(X_2)$.

²this assertion is obviously extended to the case of multivalued attributes

Bellow, we give an example of a translation table for $\phi(\mathcal{S})$ function between MADS model concepts and *SHIQ* knowledge base elements:

MASD concept	<i>SHIQ</i> expression
object types $O_1, O_2 \in \mathcal{O}_S$	$C_1, C_2 \in \mathcal{C}_K, (C_1 \sqcap C_2 \sqsubseteq \perp) \in \mathcal{T}_K$
relationship types $R_1, R_2 \in \mathcal{R}_S$	$R_1, R_2 \in \mathcal{R}_K, (R_1 \sqcap R_2 \sqsubseteq \perp) \in \mathcal{T}_K$
attribute $A \in \mathcal{A}_S$	$A \in \mathcal{R}_K$
object types hierarchy ($O_1 \text{ IsA } O_2$) $\in \preceq_S$	$(C_1 \sqsubseteq C_2) \in \mathcal{T}_K$
attribute $A \in \mathcal{A}_S$ of an object type $O \in \mathcal{O}_S$	$(\exists A.C \sqcap \forall A.C) \in \mathcal{T}_K$
cardinality constraint (min,max) on $A \in \mathcal{A}_S$	$(\exists^{\geq \text{min}} A.C \sqcap \exists^{\leq \text{max}} A.C) \in \mathcal{T}_K$
relationship type $R \in \mathcal{R}_S$, linking object types $O_1, O_2 \in \mathcal{O}_S$	$(\exists R.C_1 \sqcap \exists R^-.C_2) \in \mathcal{T}_K$
cardinality constraint (min,max) on $R \in \mathcal{R}_S$	$(\exists^{\geq \text{min}} R.C_1 \sqcap \exists^{\leq \text{max}} R.C_1) \in \mathcal{T}_K$
relationship type hierarchy ($R_1 \text{ IsA } R_2$) $\in \preceq_S$	$(R_1 \sqsubseteq R_2) \in \mathcal{T}_K$
spatial relationship type $R^{\text{spa}} \in \mathcal{R}_S^{\text{spa}}$	
linking object types $O_1, O_2 \in \mathcal{O}_S^{\text{spa}}$	$(\exists R^{\text{spa}}.C_1 \sqcap \exists R^{\text{spa}-}.C_2 \sqcap \forall R^{\text{spa}}.C_1 \sqcap \forall R^{\text{spa}-}.C_2) \in \mathcal{T}_K, C_1, C_2 \in \mathcal{C}_K^{\text{spa}}$
temporal relationship type $R^{\text{tem}} \in \mathcal{R}_S^{\text{tem}}$	
linking object types $O_1, O_2 \in \mathcal{O}_S^{\text{tem}}$	$(\exists R^{\text{tem}}.C_1 \sqcap \exists R^{\text{tem}-}.C_2 \sqcap \forall R^{\text{tem}}.C_1 \sqcap \forall R^{\text{tem}-}.C_2) \in \mathcal{T}_K, C_1, C_2 \in \mathcal{C}_K^{\text{tem}}$

Using this translation between MADS and *SHIQ* constructs we are now able to represent MADS binary relationships, cardinalities on attributes and roles, relationship and object types hierarchies that are used to define the spatial and temporal dimensions in the *SHIQ* formalism. In the following section we will show how we implement the spatial and temporal dimensions in the *SHIQ*-based language and what are the additional restrictions required for the resulting DL model in order to model MADS spatial and temporal dimensions' particularities.

5.2 Data Model Definition in OWL

In this section we present the translation of the MADS data model into the OWL language (based on the *SHIQ* description logic) suitable for reasoning with the RACER reasoner. The language that we use is the OWL-DL language. We describe the translation of the MADS structural, spatial, temporal dimensions, constrained relationships, and representation dimension. We refer to the MADS model expressed in OWL as MADS-OWL.

5.2.1 Structural Dimension.

A DL model is composed of classes, properties, axioms, and restrictions. In the terms of the MADS models, a schema is composed of object types, attributes and

relationships. If in the MADS model attributes are defined in the scope of an object type, i.e., attributes cannot exist by themselves, in a DL model, the sets of classes and properties are independent, i.e., a property can be defined in general without stating to what class(es) it belongs.

Object types definition. Let us see the simplest definition, an object type `Tram` from Figure 3.13, defined as a class in OWL:

```
< owl : Class rdf : ID = "Tram" / >
```

In the above definition we declare a `Class` (OWL syntax or namespace) with an ID (RDF namespace) equal to `Tram`. The OWL namespace inherits all the terms of the RDF syntax, and therefore where appropriate, the RDF tags are used. There are as well other namespaces that may be used the OWL schemas definitions, for example DAML tags. All allowed namespaces are explicitly declared in the header of an `.owl` file, for example:

```
swrl = "http://www.w3.org/2003/11/swrl"
rdfs = "http://www.w3.org/2000/01/rdf-schema"
owl = "http://www.w3.org/2002/07/owl"
```

For the inheritance or generalization/specialization link, a class can have subclasses, and super-classes. In the DL model, for the `Tram` object type from Figure 3.13, the equivalent expression for the DL class is:

```
< owl : Class rdf : ID = "Tram" >
  < rdfs : subclassOf >
    < owl : Class rdf : ID = "TransportLine" / >
  < /rdfs : subclassOf >
< /owl : Class >
```

A class can have several subclasses and several super-classes, this last feature allows for multi-inheritance, which is the important feature of the MADS model. The multi-inheritance in the DL interpretation does not provide for any refinement, re-definition, or overloading like in MADS model [Don02]. The subclass inherits all the properties and restrictions from its super-classes. Due to the uniqueness assumption for the terms in a DL model, this does not produce errors. As an example for the multi-inheritance, the same class `Tram` can be defined as a subclass of two classes `TransportLine` and `PowerConsumer` as follows:

```
< owl : Class rdf : ID = "Tram" >
  < rdfs : subclassOf >
    < owl : Class rdf : ID = "TransportLine" / >
```

```

    < /rdfs : subClassOf >
    < rdfs : subClassOf >
      < owl : Class rdf : ID = "PowerConsumer" / >
    < /rdfs : subClassOf >
  < /owl : Class >

```

MADS supports multi-instantiation, which means that several instances in different types may represent the same real-world phenomenon, which in turn materializes in the fact that such related instances bear the same identity. This originates from the traditional hypotheses of object-oriented databases, in particular the uniqueness of the most specialized instance for an object in an *is-a* hierarchy allowing an object to be simultaneously represented in several classes. MADS proposes two multi-instantiation links: the *is-a* link and the *may-be* link. In OWL, two classes may contain common instances, even if they are not linked by an *is-a* link. In MADS, by default two object (or relationship) types with no common ancestor in the generalization hierarchy are disjoint. To impose the same behavior on the classes in a DL model, the designer must state explicitly the disjointness axiom for two or more classes. This will assure (by the reasoner) that neither of the two classes would be allowed to have any instances in common. In the example we state that the class *River* is disjoint with the class *Rails*:

```

  < owl : Class rdf : ID = "River" >
    < owl : disjointWith >
      < owl : Class rdf : ID = "Rails" / >
    < /owl : disjointWith >
  < /owl : Class >

```

For the schema in Figure 3.13, the above statement is derived from the disjointness axiom imposed on their ancestors.

The covering axiom for subclasses in MADS is translated in a DL model by the `owl:unionOf` axiom. The `owl:unionOf` axiom in a class description defines this class as a subclass of an anonymous class defined in turn as the union of the several other classes. An `owl:unionOf` axiom describes an anonymous class for which the class extension contains those individuals that occur in at least one of the class extensions of the class descriptions in the list. Moreover, this axiom implements the closed world assumption for the involved class which is the default assumption for the database world (contrary to the logics world). If we look for an example illustrating the covering axiom, the *TouristSite* class from Figure 3.13 can only have instances of classes *Museum*, *Monument*, *Theatre*, or *Walk*:

```

  < owl : Class rdf : about = "TouristSite" >
    ...
    < rdfs : subClassOf >
    < owl : Class >

```

```

    < owl : unionOf rdf : parseType = "Collection" >
      < owl : Class rdf : ID = "Monument" / >
      < owl : Class rdf : ID = "Museum" / >
      < owl : Class rdf : ID = "Theatre" / >
      < owl : Class rdf : ID = "Walk" / >
    < /owl : unionOf >
  < /owl : Class >
< /rdfs : subclassOf >
...
< /owl : Class >

```

With the described above features of the DL language, we can define object types, the generalization/specialization links with disjoint and cover axioms, and multi-inheritance as in the MADS model. In the next section we show the correspondence between the relationships in MADS and properties in DL.

Relationship types definition. A MADS relationship is a link between two or more object types, where each object type is assigned a role. A relationship type describes a set of links with similar characteristics, i.e., linked objects are of the same type, with the same roles, and similar properties. An OWL property is a binary relation that connects two OWL concepts, where the linked concepts can be either classes, or a class and a datatype; a property has a range and a domain. The domain of a property is a class this property is defined in; domain can be either a user defined class or several classes, or the `owl:Thing` which is the root class for a model. If a property defined with the `owl:Thing` as its domain, then by inheritance, all classes in the model will have this property. Depending on the range of the property, the property can be either a object property or datatype property. Datatype properties link an individual from a class (defined by the domain of this property) to an XML schema datatype value or an RDF literal. Object properties link an individual from a class to another individual from another (linked by this property) class. To represent MADS relationships we employ OWL object properties. The datatype properties correspond to simple attributes in MADS and they are considered in the sequel of this section. In addition to the object and datatype properties, OWL provides for annotation properties. Annotation properties can be used to add metadata to classes, individuals, object, or datatype properties.

To define a MADS relationship `tramBy` type (Figure 3.13) the following OWL code is written:

```

< owl : ObjectProperty rdf : ID = tramBy" >
  < rdfs : domain rdf : resource = "Tram" / >
  < rdfs : range rdf : resource = "Rails" / >
< /owl : ObjectProperty >

```

If there is no cardinality restrictions, the definition of the relationship `tramBy` is completed. In the schema in Figure 3.13, there is a cardinality 1:n imposed on the role `Tram-tramBy`, this cardinality is translated in the OWL restriction clause. The following description of the class `Tram` includes the cardinality constraints 1:n on the role `Tram-tramBy`:

```
< owl : Classrdf : ID = "Tram" >
  < rdfs : subclassOf >
  < owl : Restriction >
  < owl : onProperty >
    < owl : ObjectProperty rdf : ID = "tramBy" / >
  < /owl : onProperty >
  < owl : minCardinality
    rdf : datatype = "http://www.w3.org/2001/XMLSchema int" > 1
  < /owl : minCardinality >
  < /owl : Restriction >
  < /rdfs : subclassOf >
```

The above restriction makes the property mandatory, i.e., having at least one value. OWL provides three constructs for restricting the cardinality of properties locally within a class context. Besides the `owl:minCardinality`, there is the `owl:maxCardinality` restricting the maximum number of values, and the `owl:Cardinality` if the property must have an exact number of distinct values. With these constructs, all types of MADS cardinality constraints can be translated to OWL syntax.

OWL object properties are unidirectional by default. To define a relationship as being bidirectional as in MADS, an inverse property must be defined. For example, for the relationship `along` between the object types `TransportLine` and `Stop` from Figure 3.13, we define a property `along` with the domain `TransportLine` and the range `Stop`, and then create an inverse property `inverse_of_along` switching the domain and the range:

```
< owl : ObjectProperty rdf : ID = "inverse_of_along" >
  < rdfs : range rdf : resource = "TransportLine" / >
  < owl : inverseOf >
    < owl : ObjectProperty rdf : ID = "along" / >
  < /owl : inverseOf >
  < rdfs : domainrdf : resource = "Stop" / >
  < /owl : ObjectProperty >
```

Besides binary relationships, in MADS n -ary relationships that link more than two object types are allowed. An example of such a relationship is the relationship `along` in Figure 3.10. It links three object types `Park`, `Non-Planted Area` and `Field`, where `Non-Planted Area` could be either a built-up area, a water body, or a road.

There is no direct way to model the same schema part in OWL. Though, a particular case of an n -ary relationship can be modeled, i.e., when the cardinalities of all the roles are 1:1. Let us describe this particular case assuming that the cardinalities of the roles of the relationship `along` are 1:1. The relationship `along` is then defined as an object property with domain and range set to the disjunction of the three classes it links. To each of the classes we then add the cardinality restriction on the role `along` equal to the double of the original cardinality thus, we require 2 instances in the range of the property. The interpretation of the ternary relationship `along`³ in Figure 3.10 is, for each instance of the `Park` there exist two instances linked to it, one of the type `Non-Planted Area` and a second one of type `Field`. Note, if the cardinalities of this relationship are different than 1:1 we cannot say how many instances are related by each instance of this relationship, and this would make it impossible to define the restriction over the property `along` in OWL. To ensure that an instance of the class `Park` is linked by the property `along` to exactly one instance of the class `Non-Planted Area` and exactly one instance of the class `Field`, we add the existential restrictions for the latter two classes in the former class:

```
<owl:Class rdf:ID="Park">
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty>
        <owl:ObjectProperty rdf:ID="along"/>
      </owl:onProperty>
      <owl:cardinality rdf:datatype="int">2</owl:cardinality>
    </owl:Restriction>
  </rdfs:subClassOf>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:someValuesFrom rdf:ID="NonPlantedArea"/>
      <owl:onProperty>
        <owl:ObjectProperty rdf:ID="along"/>
      </owl:onProperty>
    </owl:Restriction>
  </rdfs:subClassOf>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:someValuesFrom rdf:ID="Field"/>
      <owl:onProperty>
        <owl:ObjectProperty rdf:ID="along"/>
      </owl:onProperty>
    </owl:Restriction>
  </rdfs:subClassOf>
  ...
</owl:Class>
```

³assuming that the cardinalities are 1:1 for all the roles.

These two existential restrictions together with the property cardinality set to 2, model the phenomenon of three linked instances. Similar restrictions are added to other two classes, **NonPlantedArea** and **Field**. The last restriction that must be added to the model, is an integrity constraint to guarantee that the linked instances are the same, Figure 5.2. Another way to model an n -ary relationship is to decompose it in n

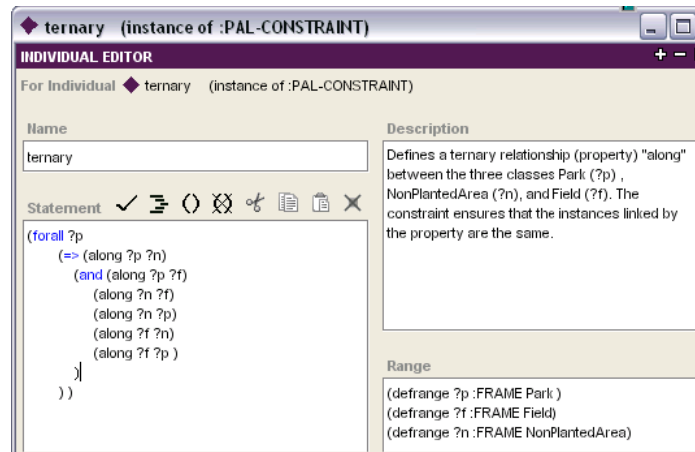


Figure 5.2: Integrity constraint for the ternary relationship *along* from Figure 3.10

binary relationships as described for example in Chapter 3 "The Entity-Relationship Model" of [Tha00], figure 5.3 illustrates the decomposition for the same schema part with the initial cardinality constraints.

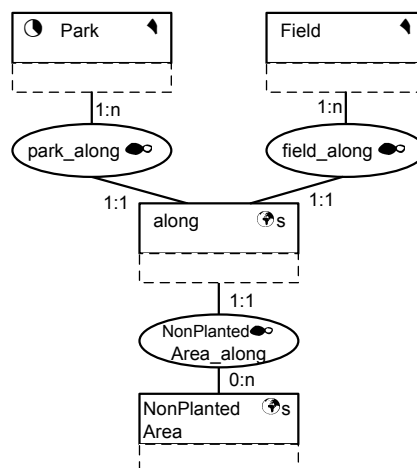


Figure 5.3: Decomposed ternary relationship *along* from Figure 3.10

Like for the object types, in MADS we can define the generalization/specialization link between relationship types. OWL also support hierarchies of properties

which can be specified with `owl:subPropertyOf`. For example, the relationship `along`, is a topological relationship with the inclusion semantics. To define this type relationships in OWL we first create a basic topological relationships `include` (this is detailed in the sequel of this section), and then define the `along` as a sub-property of the `include`:

```
< owl : ObjectProperty rdf : ID = "along" >
  < rdfs : domain rdf : resource = "TransportLine" / >
  < rdfs : subPropertyOf rdf : resource = "include" / >
  < owl : inverseOf rdf : resource = "inverse_of_along" / >
  < rdfs : range rdf : resource = "Stop" / >
< /owl : ObjectProperty >
```

The discussion above concerns the translation of the relationships that hold between two instances. In the MADS model, there is another type of association called multi-association (cf. Section 3.2.5) defined for linking sets of instances through a single relationship link. For this kind of relationships there are no corresponding OWL construct. This binary restriction for properties limits OWL model expressiveness as compared to the MADS model. The aggregation link falls into the range of constructs that are not directly represented in DL based models. An aggregation link is a relationship that conveys a specific semantics of composite and component object types, where a composite instance is an aggregation of component instances. The aggregation links are translated into OWL by object properties with predefined cardinality constraints and quantifier restrictions. For semantic clarity, using predefined names for these properties would be beneficial, but due to the unique name assumption, we cannot use the same name for different properties. The following code describes class `City` from Figure 3.12, that has a property `decomposed` with the range `District`. To express the aggregation link `composed` between `City` and `District` we add the `owl:minCardinality` constraint to the property to ensure that there is at least one component instance; and we add the universal quantifier restriction to ensure the closed world assumption and the membership for all the component instances.

```
< owl : Classrdf : ID = "City" >
...
< rdfs : subclassOf >
  < owl : Restriction >
    < owl : onProperty >
      < owl : ObjectProperty rdf : ID = "decomposed" / >
    < /owl : onProperty >
    < owl : minCardinality
      rdf : datatype = "http : //www.w3.org/2001/XMLSchema int > 1"
    < /owl : minCardinality >
  < /owl : Restriction >
```

```

</rdfs:subClassOf>
<rdfs:subClassOf>
  <owl:Restriction>
    <owl:allValuesFrom>
      <owl:Class rdf:ID="District"/>
    </owl:allValuesFrom>
    <owl:onProperty>
      <owl:ObjectProperty rdf:ID="decomposed"/>
    </owl:onProperty>
  </owl:Restriction>
</rdfs:subClassOf>
</owl:Class>

```

Using the DL based modeling we cannot express the aggregation semantic, i.e., we cannot say that the same phenomenon in our example, a city and a set of districts, in the real world is the same object. The `owl:sameClassAs` construct allows for such statements only between two instances but not between an instance and a set of instances. The whole-part relationship is partially modeled in OWL with the transitivity axiom imposed on an object property. But the limitation of this axiom is that OWL-DL requires that for a transitive property no local or global cardinality constraints should be declared on the property itself or its super-properties, nor on the inverse of the property or its super-properties.

Finally, the last structural element that is used in MADS and in general in conceptual models, is an attribute. In MADS, both object types and relationship types may hold attributes. Depending on the data model, attributes can be either complex or simple, mono-valued or multi-valued, mandatory or optional. In the sequel we consider the translation of the MADS attributes into the OWL language.

Attributes definition. MADS attributes are translated to OWL properties. As we mentioned above, there are two types of properties in OWL: object properties that link individuals to individuals; and datatype properties that link individuals to data values. To define a MADS attribute in OWL, new properties and classes may be needed for complex attributes as well as for their components.

Properties are defined by property axioms which define their characteristics. In its simplest form, a property axiom just defines the existence of a property. For example:

```

<owl:ObjectProperty rdf:ID="locatedIn"/>

```

Often, property axioms define additional characteristics of properties. OWL supports the following constructs for property axioms:

- **RDF Schema constructs:** `rdfs:subPropertyOf`, `rdfs:domain` and `rdfs:range`;

- **relations to other properties:** `owl:equivalentProperty` and `owl:inverseOf`
- **global cardinality constraints:** `owl:FunctionalProperty` and `owl:Inverse FunctionalProperty`;
- **logical property characteristics:** `owl:SymmetricProperty` and `owl:Transitive Property`.

To represent simple properties, like attribute `Description` of the object type `Museum` from Figure 3.13, we create an OWL property `Description`, with the domain restricted to the class `Tram`, and the range of the datatype `string`. This attribute is a mono-valued mandatory attribute, thus we add the cardinality restriction so that this property must have exactly one value for every instance of the class `Museum`. The listing shows the definition of the attribute `Description`:

```
< owl : DatatypeProperty rdf : ID = "description" >
  < rdfs : domain rdf : resource = "Museum" / >
  < rdfs : range
    rdf : resource = "http : //www.w3.org/2001/XMLSchema string" / >
< /owl : DatatypeProperty >
```

The cardinality restriction for the property `Description` in class `Museum`:

```
< owl : Restriction >
  < owl : cardinality
    < rdf : datatype = "http : //www.w3.org/2001/XMLSchema int = 1"
  < /owl : cardinality >
< owl : onProperty >
  < owl : DatatypeProperty rdf : ID = "description" / >
< /owl : onProperty >
< /owl : Restriction >
```

If a property is allowed to have a unique value, i.e., in the set of the instances of a class, there is no two instances that have the same value of a given property, than this property can be modeled as a functional property with the `owl:FunctionalProperty` axiom. Making analogy with the databases world, a functional property is an identifier attribute. Thought it is not a very realistic assumption for the above example, museum descriptions are allowed to be equal for different museums, and thus, we rather employ the cardinality restriction than the functional property axiom.

Attribute `Exhibition` of the same object type `Museum` is an optional, multi-valued attribute. To define this attribute in OWL, it is sufficient to declare datatype property `Exhibition` existence, to restrict the domain for this property to the class `Museum`, and to set the range of the property to `string`. In OWL, properties by default are optional, multi-valued properties.

Definition of a complex attribute in OWL requires additional classes to be created. In our example object type `Museum`, there is a complex attribute `OpenTime` with two component attributes `summer` and `winter`. To translate MADS attribute `OpenTime` into OWL property `OpenTime`, we first create a class `MuseumOpenTime`

```
< owl : Classrdf : ID = "MuseumOpenTime" / >
```

and then two object properties `summer` and `winter` with the domain restricted to this class.

```
< owl : ObjectPropertyrdf : ID = "summer" >
  < rdfs : domain rdf : resource = "MuseumOpenTime" / >
  < rdfs : range rdf : resource = "Interval" / >
< /owl : ObjectProperty >
< owl : ObjectProperty rdf : ID = "winter" >
  < rdfs : range rdf : resource = "Interval" / >
  < rdfs : domain rdf : resource = "MuseumOpenTime" / >
< /owl : ObjectProperty >
```

In Figure 3.13, the `winter` and `summer` are the temporal properties of type `Interval`, we do not explain here how we add the temporal features to these attributes, interested reader is referred to the section devoted to the MADS temporal dimension translation, Section 5.2.3.

Now we can create the object property `OpenTime` with domain `Museum` and range `MuseumOpenTime`.

```
< owl : ObjectProperty rdf : ID = "openTime" >
  < rdfs : rangerdf : resource = "MuseumOpenTime" / >
  < rdfs : domainrdf : resource = "Museum" / >
< /owl : ObjectProperty >
```

Having introduced the auxiliary classes and properties we have modeled in OWL complex MADS attribute `openTime`. Each value of the property `openTime` would be an instance of a class `MuseumOpenTime` with two temporal values for `winter` and `summer` properties.

As seen from the translation procedure for MADS relationship types and complex attributes, the OWL constructs we use to represent them are the same, i.e., object properties. If we consider reverse translation of an OWL model into a MADS schema, then these two different MADS constructs become indistinguishable. We propose to employ the annotation property to label MADS complex attributes. OWL DL allows annotations on all the basic constructs, i.e., classes, properties, individuals. Five annotation properties are predefined by OWL, `owl:versionInfo`, `rdfs:label`, `rdfs:comment`, `rdfs:seeAlso`, `rdfs:isDefinedBy`. We use the `rdfs:comment` with

the value *"MADS complex attribute"* associated to the object property that models an MADS complex attribute. Finally, the translation of the `openTime` attribute is as follows:

```
< owl : ObjectProperty rdf : ID = "openTime" >
  < rdfs : comment
    rdf : datatype = "http : //www.w3.org/2001/XMLSchema string"
    > MADS complex attribute < /rdfs : comment >
  < rdfs : range rdf : resource = "MuseumOpenTime" / >
  < rdfs : domain rdf : resource = "Museum2" / >
< /owl : ObjectProperty >
```

As in MADS, OWL provides the construct for defining a range of data values, namely an enumerated datatype. This datatype format makes use of the `owl : oneOf` construct, that is also used for describing an enumerated class. In Figure 3.13 there is an attribute `season` in the object types `Boat` and `Walk`. This attribute can have from 1 to 4 predefined values for season names. An OWL corresponding property would be an enumerated datatype property with the allowed values from the list `summer`, `autumn`, `winter`, and `spring`. The listing shows the OWL constructs for this property.

```
< owl : DatatypeProperty rdf : ID = "season" >
  < rdfs : range >
  < owl : DataRange >
  < owl : oneOf >
    < rdf : List >
      < rdf : first
rdf : datatype = "http : //www.w3.org/2001/XMLSchema string" > spring
      < /rdf : first >
      < rdf : rest >
      < rdf : List >
        < rdf : first
rdf : datatype = "http : //www.w3.org/2001/XMLSchema string" > summer
        < /rdf : first >
        ...
      < /rdf : List >
      < /rdf : rest >
    < /rdf : List >
  < /owl : DataRange >
< /rdfs : range >
```

We have described the representation of the MADS object type attributes in OWL. In MADS, not only object types but also the relationship types can hold attributes. Since both MADS relationships and attributes are represented by the

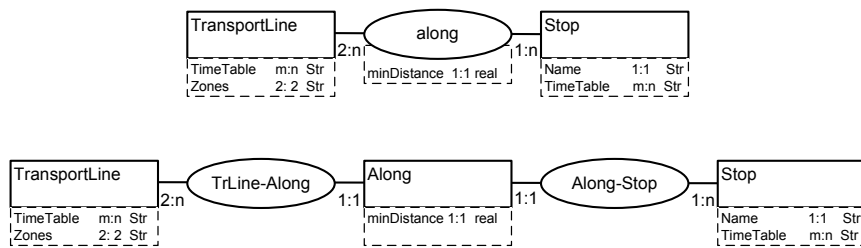


Figure 5.4: Transformation for relationships with attributes.

same OWL construct, there is no direct way to represent such kind of attributes in OWL. MADS relationships holding attributes should be translated to object types with attributes. To keep the link with the object types that were related by this relationship, two more relationship types should be added, as shown in Figure 5.4.

Let us summarize all the structural elements then we can use in the two modeling approaches - MADS and OWL, the Table 5.1 lists the concepts that we have described in this section.

Table 5.1: Structural Dimension - MADS vs OWL

MADS concepts	OWL concepts
Object type	<code>owl:Class</code>
IsA link	<code>owl:SubClassOf</code>
Covering axioms: cover, disjoint	<code>owl:UnionOf</code> , <code>owl:DisjointWith</code>
no corresponding concept	<code>owl:sameClassAs</code>
no corresponding concept	Enumerated class <code>owl:OneOf</code>
Binary relationship type	<code>owl:ObjectProperty</code> with defined <code>owl:range</code> , <code>owl:domain</code> , and <code>owl:inversePropertyOf</code>
n -ary relationship type	no direct representation
IsA link	<code>owl:subPropertyOf</code>
Role cardinalities	<code>owl:minCardinality</code> , <code>owl:maxCardinality</code> <code>owl:Cardinality</code>
Object simple attribute	<code>owl:datatypeProperty</code>
Object complex attribute	<code>owl:objectProperty</code>
Identifier attribute	<code>owl:functionalProperty</code>
no corresponding concept	Relations to other properties <code>owl:equivalentProperty</code>
Relationship attribute	no direct representation

5.2.2 Spatial Dimension.

As we have described in Section 3.2, the spatial dimension of the MADS model contains the hierarchy of spatial abstract data types (SADT) and the set of topolog-

ical relationships with associated constraints. The SADTs are associated to object types and attributes to add specific spatial features whereas, topological features are added to relationship types. The spatial dimension in MADS is orthogonal to the structural dimension. Similarly for OWL modeling, we aim at defining the spatial dimension in a way it could be freely added or removed as additional feature to structural elements. The general idea is to create a hierarchy of OWL classes with associated restrictions, and use the members of this spatial hierarchy as subclasses for those user defined classes to which we want to associate spatial features. With the `owl:subClassOf` construct, the restrictions of the superclass are inherited by the subclasses, preserving the spatial behavior in the subclasses.

The definition of the spatial domain starts with the definition of two disjoint classes `GSimple` and `GComplex`:

```
< owl : Class rdf : ID = "GSimple" >
  < owl : disjointWith rdf : ID = "GComplex" / >
< /owl : Class >
< owl : Class rdf : ID = "GComplex" >
  < owl : disjointWith rdf : ID = "GSimple" / >
< /owl : Class >
```

or graphically in the Protégé editor as shown in Figure 5.5⁴. Instances of the class `GSimple` are the `point`, `line`, `lineSetOriented`, and `areaSimple`. Instances of the class `GComplex` are `pointSet`, `lineSet`, `lineSetOriented`, and `areaComplex`. These instances

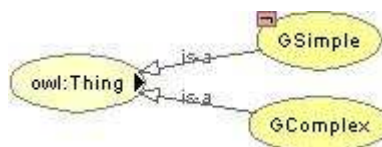


Figure 5.5: OWL: Geo types for basic spatial instances.

are the basic elements to define the spatial dimension in OWL. The intrinsic property of a spatial class is the `hasGeometry` property. The `hasGeometry` property for different spatial types has different predefined values, those values are either values from class `GSimple` or class `GComplex`. Therefore, we define a general (or root) spatial type `Geo`, and restrict the domain of the property `hasGeometry` to this class, and the range to the union of classes `GSimple` and `GComplex`.

```
< owl : ObjectProperty rdf : ID = "hasGeometry" >
  < rdfs : range >
    < owl : Class >
```

⁴As the class definitions become more complex, we will use graphical notation instead of the textual one.


```

    < owl : unionOf rdf : parseType = "Collection" >
      < owl : Class rdf : about = "GSimple" / >
      < owl : Class rdf : about = "GComplex" / >
    < /owl : unionOf >
  < /owl : Class >
< /rdfs : range >
< rdfs : domain >
  < owl : Class >
    < owl : unionOf rdf : parseType = "Collection" >
      < owl : Class rdf : about = "Geo" / >
    < /owl : unionOf >
  < /owl : Class >
< /rdfs : domain >
< /owl : ObjectProperty >

```

As the listing above shows, we have defined the `hasGeometry` property with the closed world assumption for its range, i.e., values of this property are limited only to the predefined instances of two classes. Moreover, by applying the closed world assumption to the domain of the `hasGeometry` property this property becomes an intrinsic property for spatial classes. We all all the restrictions on the `hasGeometry` property to the set of *necessary & sufficient* conditions to define (contrary to describe) the sub-class of the `Geo` class. With such a set of *necessary & sufficient* conditions, any class that has the `hasGeometry` property will by classified by the reasoner as a spatial class.

Now we can define the OWL spatial hierarchy by extending the root `Geo` class and stating the values for the `hasGeometry` property for each spatial subclass. The spatial hierarchy is shown in Figure 5.6. The two direct subclasses of the `Geo` are disjoint therefore, all their subclasses are as well disjoint. For the `GeoSimple` class the `hasGeometry` property has the range `GSimple` and, for the `GeoComplex` class the `hasGeometry` property has the `GComplex` class as its range. Then, for the leaf classes, the `hasGeometry` property has an exact value from the corresponding class. For example, for `LineSet` class, the restriction of the `hasGeometry` property is the following:

```

< owl : Class rdf : ID = "LineSet" >
  < owl : equivalentClass >
    < owl : Restriction >
      < owl : onProperty >
        < owl : ObjectProperty rdf : ID = "hasGeometry" / >
      < /owl : onProperty >
      < owl : hasValue >
        < GComplex rdf : ID = "lineSet" / >
      < /owl : hasValue >
    < /owl : Restriction >
  < /owl : equivalentClass >

```

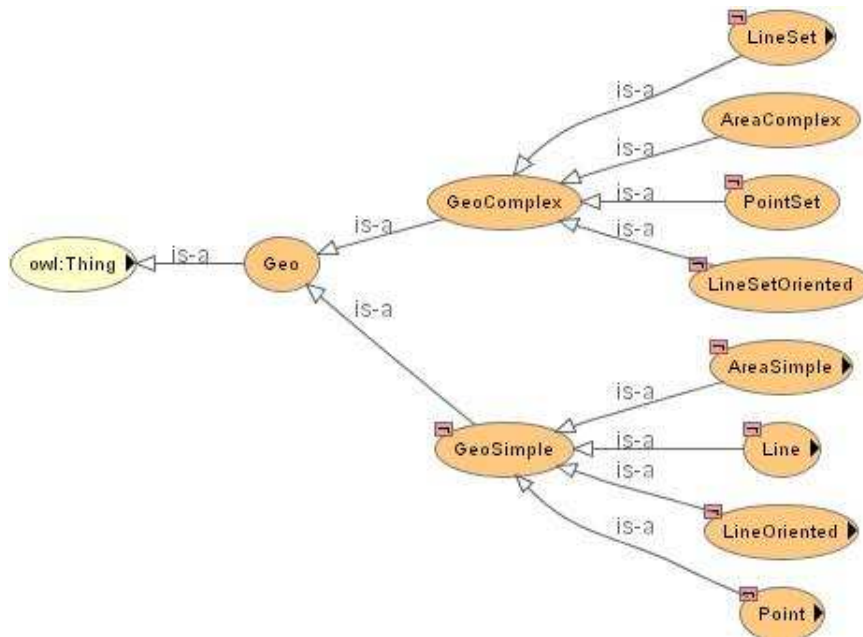


Figure 5.6: OWL: spatial data types hierarchy.

...

The listing above instructs the reasoner to classify any class and any instance that has the property `hasGeometry` with the value `lineSet` of type `GComplex`, as being of type `LineSet`.

By defining the hierarchy of the spatial types and restrictions on the intrinsic property `hasGeometry`, we have created the spatial dimension for the OWL models. The spatial classes we defined are abstract, i.e., there will be no direct instances of the class `Line` or any other class from the hierarchy created. Instead, we use the spatial types as super-classes for user defined classes. The type hierarchy has some limitations compared to the MADS spatial data type hierarchy. We had to introduce two additional classes and populate them with specific instances that we later used to distinguish the spatial types. Using instances instead of classes limits the rules over the spatial types that we were able to express. This drawback is due to the fact that we use the OWL-DL language. In OWL-Full we would be able to use directly a class for lines, a class for points etc. But since the OWL-Full is not optimized for reasoning, we allow for the OWL-DL limitations. Using instances instead of classes has some drawbacks. We cannot say that for example, a `lineOriented` is a subtype of the `line`. Later we will see the consequences of the lack of the `IsA` link between these types. We summarize the two spatial dimensions in the Table 5.2.

Table 5.2: Spatial Dimension - MADS vs OWL

MADS spatial dimension	OWL spatial dimension
Spatial ADT hierarchy	Spatial abstract types hierarchy
Hierarchy in spatial subclasses	not possible in OWL-DL
Mandatory Geometry attribute that defines spatial features	Intrinsic hasGeometry property with predefined values

5.2.3 Temporal Dimension.

The temporal dimension is defined in a similar way to the spatial dimension. First, we create two disjoint classes, **TComplex** and **TSimple**:

```
< owl : Class rdf : ID = "TSimple" >
  < owl : disjointWith rdf : ID = "TComplex" / >
< /owl : Class >
< owl : Class rdf : ID = "TComplex" >
  < owl : disjointWith rdf : ID = "TSimple" / >
< /owl : Class >
```

According to the MADS temporal abstract type hierarchy shown in Figure 3.5, we create the populations of these two classes, i.e., **instant** and **interval** are the instances of class **TSimple**; and **instantSet**, **intervalSet** are the instances of the class **TComplex**.

The intrinsic property for the temporal types is the **hasTime** property. This means that the necessary and sufficient condition for a class to be classified as a temporal one, is the existence of the **hasTime** property. For the general (or root) temporal class **Time**, this property must exist, and the value of this property is restricted to one of the values from the **TSimple** or **TComplex** classes.

```
< owl : Class rdf : ID = "Time" >
  < owl : equivalentClass >
  < owl : Restriction >
    < owl : someValuesFrom >
      < owl : Class >
        < owl : unionOf rdf : parseType = "Collection" >
          < owl : Class rdf : ID = "TComplex" / >
          < owl : Class rdf : ID = "TSimple" / >
        < /owl : unionOf >
      < /owl : Class >
    < /owl : someValuesFrom >
  < owl : onProperty >
    < owl : ObjectProperty rdf : ID = "hasTime" / >
```

```

    </owl:onProperty >
    </owl:Restriction >
    </owl:equivalentClass >
  </owl:Class >

```

Defined this way, a temporal class must have the `hasTime` property. The closure axiom is applied for the subclasses of the `Time` class. The temporal hierarchy for the OWL models is shown in Figure 5.7. For the subclasses we define additional

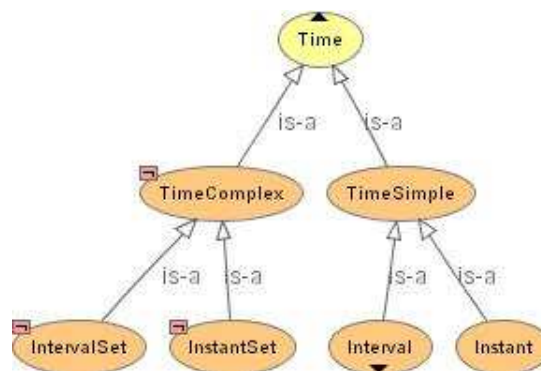


Figure 5.7: OWL: temporal data types hierarchy.

axioms on the values of the property `hasTime`. For example, let us consider in details the temporal type `Interval`, its representation in the Protégé tool is shown in Figure 5.8. All the restrictions concern the `hasTime` property. From the root class `Time`, `Interval` inherits the most general restriction on the `hasTime` property. Then, from the `TimeSimple` class a more precise restriction is inherited. This restriction is compliant to the previous, more general one, and it restricts the range of the `hasTime` property only to the instances of the `TSimple` class. Finally, in the leaf class `Interval`, the restriction on the `hasTime` property is most precise, the necessary and sufficient condition for a class to be classified as `Interval`, is that the value of the intrinsic property must be equal to `interval`, where `interval` is one of the instances of class `TSimple`. As seen from restriction inherited from the `TSimple` type (Figure 5.8), we have imposed the closure axiom on the values of the `hasTime` property. This axiom allows for reasoning with the closed world assumption, which is the default assumption in the database world.

We show in Table 5.3 the MADS and OWL structures that are used to model temporal dimension in our approach. The temporal ADT hierarchy in MADS model has only two levels, which makes its translation to the OWL language simpler. For such hierarchy we could specify all the rules that exist for the initial MADS TADT hierarchy.

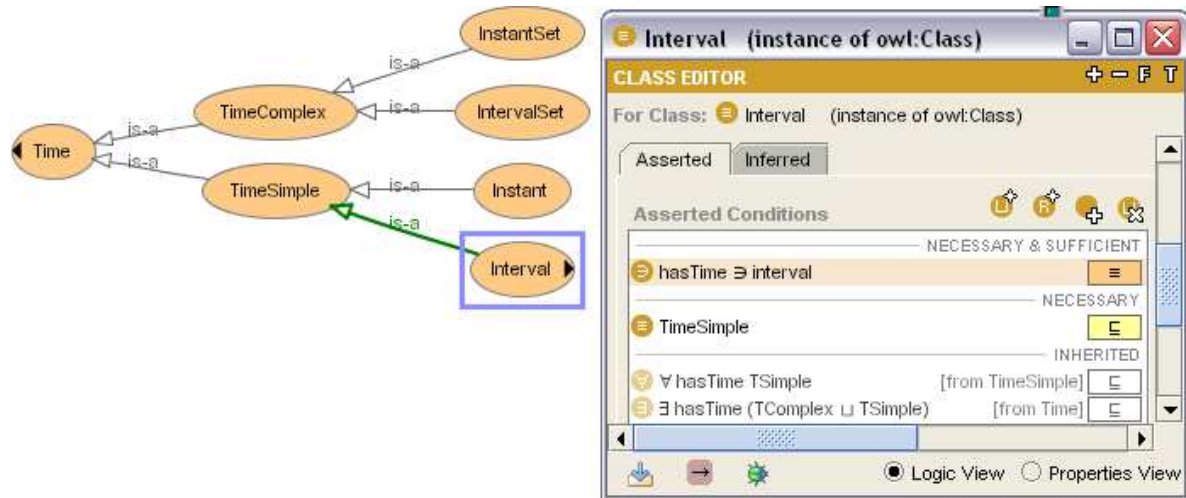


Figure 5.8: OWL: Interval temporal type.

Table 5.3: Temporal Dimension - MADS vs OWL

MADS temporal dimension	OWL temporal dimension
Temporal ADT hierarchy	Temporal abstract types hierarchy
Mandatory LifeCycle attribute that defines spatial features	Intrinsic hasTime property with predefined values

5.2.4 Constrained relationships.

We have introduced MADS constrained relationships in Section 3.2.4. Two tables 3.2 and 3.3 list the topological and synchronization relationships supported in the MADS data model. These relationships hold between a limited subset of MADS object types: topological relationships hold between spatial object types, and synchronization relationships hold between temporal object types.

Topological relationships hold between object types featuring spatial properties. To implement this kind of relationships in the OWL language, we introduce a set of OWL object properties with associated restrictions. These restrictions ensure that firstly, the properties hold only between spatial classes and secondly, that the types of the related classes correspond to the type of the topological property, i.e., for each particular topological property, there are subsets of spatial classes that are allowed as its range and domain. We show possible topological relationships in Figure 5.9.

As the figure shows, the distribution of the topological properties over the data types is not regular, and thus, we will define them differently. To define the *disjoint* property, we add this property to the definition of the spatial dimension with the domain and range set to *Geo* because all pairs of spatial types can participate in the *disjoint* property. Then, we defined properties that form segments in the validity table, e.g., the *meet* property holds for all line and area types. For the the *adjacent*

domain \ range	•	•••	~	×	◀▶
•	•	•°	•°	•°	•°
•••	•°	•°	•°	•°	•°
~	•°	•°	•°	•°	•°
×	•°	•°	•°	•°	•°
◀▶	•°	•°	•°	•°	•°

•°	•°	•°	•°	•°	•°
----	----	----	----	----	----

•° disjoint •° meet •° overlap •° cross •° include •° equal

Figure 5.9: MADS: topological relationships, validity table.

topological property the domain and range composed of the `Line`, `LineOriented`, `LineSet`, `LineSetOriented`, `AreaSimple`, and `AreaComplex` spatial types. Figure 5.10 shows the OWL representation of the MADS adjacent topological relationship.

The `cross` property holds for the sub-table that excludes the `Point` spatial type, the pairs `PointSet-PointSet`, and both simple and complex areas. In MADS, two point sets, and two areas may overlap but not cross [Don02]. Due to this notion of the `cross` relationship, we define the `cross` property separately for sets of points, lines, and areas. According to the table depicted in Figure 5.9, for the point set spatial type, the property `cross_point` is defined with the domain of `PointSet` and the range of line and area spatial types. For line spatial types we define the `cross_line` property with the domain `Line`, `LineOriented`, `LineSet`, and `LineSetOriented` and the range composed of point set spatial type and all line and area spatial types. Then, we define the `cross_area` property with the domain containing area spatial types and the range containing point set and line spatial types.

Following the same idea, we define topological inclusion in OWL. The property `include` is defined separately for point set, line, and area spatial types, the domain and range are set in correspondence to the validity table for the inclusion topological relationship. The rule for the inclusion topological property is that the domain of this property may contain one- or two-dimensional spatial classes, and range should contain spatial classes with the lesser or equal dimension. For example, a property that has the line types in its domain, will have point and line types in its range. The spatial inclusion property with the area types in its domain, will have all other spatial classes in its range, because areas are two-dimensional classes, the maximal dimension supported in MADS model. For our OWL spatial dimension we define three inclusion properties with point, line and area spatial types as their domains and the ranges set according to the rule described above.

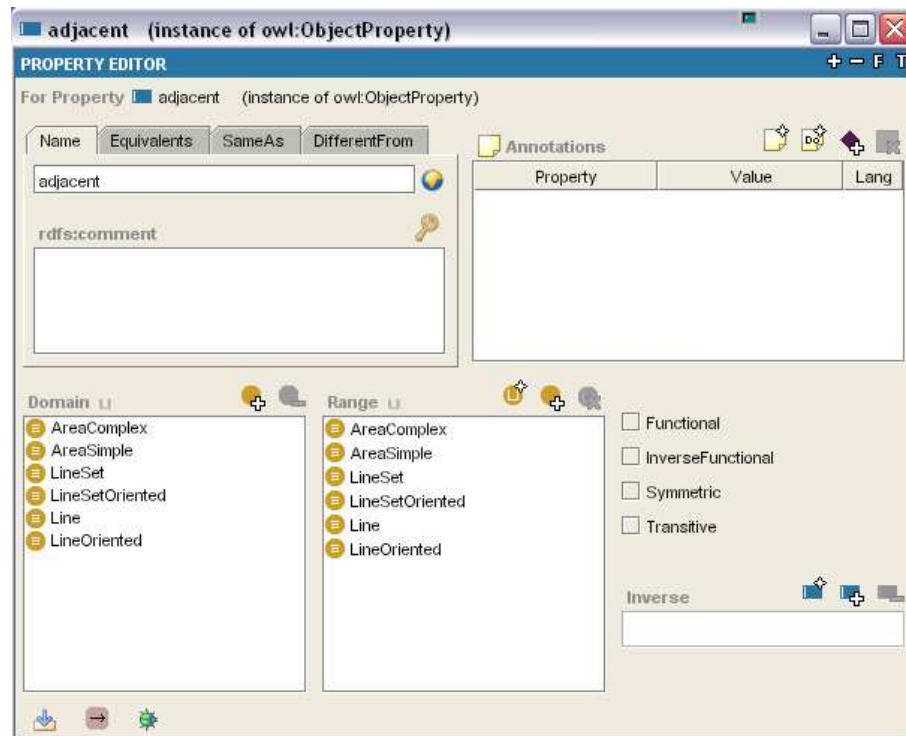


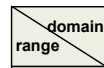

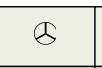



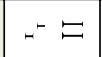
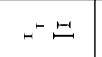
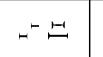

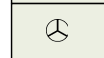
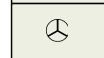
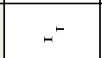
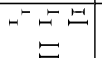
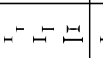
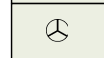
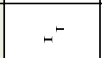
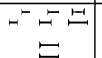
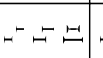



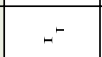
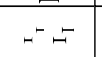
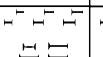

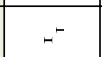
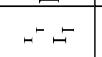
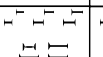

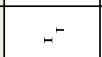
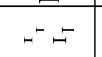
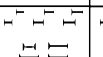



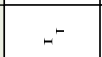
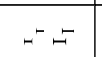
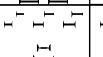

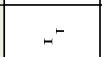
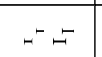
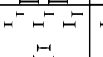

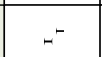
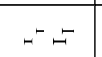
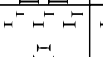
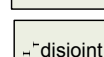
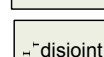
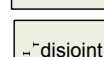
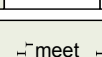
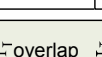
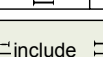
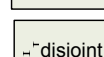
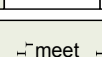
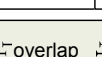
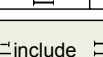
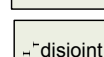
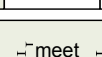
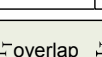
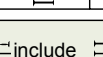
Figure 5.10: OWL: adjacent topological property.

The domain and range of the `overlap` property are composed of the classes with equal dimensions but it is not symmetrical, i.e., an `Area` can be the range for an `overlap` property with the domain `AreaComplex`. For the `overlap` MADS relationship, we define three different properties in OWL: `overlap_pointSet`, `overlap_line`, and `overlap_area`.

The distribution of the `equal` topological property has a different pattern. This property is a symmetrical one, i.e., its domain and range are always the same. Instead of defining `equal` property for each of eight spatial types, we define a generic property `equal`, and then for each spatial type we restrict this property by a universal restriction. For example, for the `PointSet` spatial type, we add the following restriction on the `equal` property:

```
< owl : Class rdf : ID = "PointSet" >
  < rdfs : subclassOf >
    < owl : Restriction >
      < owl : onProperty rdf : resource = "equal" / >
      < owl : allValuesFrom rdf : resource = "PointSet" / >
    < /owl : Restriction >
  < /rdfs : subclassOf >
< /owl : Class >
```

Let us now describe the relationships between temporal classes. These relationships are called synchronization relationships in MADS. To define these relationships in OWL, we introduce synchronization properties for our MADS-OWL model. To define the OWL synchronization properties, we construct the validity table in the same way we did for topological relationships, the table is shown in Figure 5.11; the synchronization properties in MADS-OWL are defined in correspondence to it.

domain range				
	    		   	   
		   	   	   
		   	   	   
		   	   	   






    

Figure 5.11: MADS: synchronization relationships, validity table.

Similarly to the topological properties, the domain and range for a synchronization property are temporal classes. This restriction guarantees that synchronization properties belong to the temporal extension of the model. Making further analogy with the spatial dimension, we note that synchronization relationships correspond to the topological relationships between point and line spatial types. This is because the temporality of a class is expressed by associating an instant (point in time), interval (line in time), set of instants, or set of interval type to it, which corresponds in the spatial dimension, to point, oriented line, set of points, and set of oriented lines respectively.

The relationship that exists for all temporal types is **disjoint**. For our MADS-OWL model, we have defined the **tdisjoint** property⁵ with two sub-properties **before** and **after**; the screen-shot in shown in Figure 5.12. The **tdisjoint** property has its domain and range set to **TimeSimple** or **TimeComplex** temporal classes. The sub-properties **before** and **after**, inherit these restrictions. For every temporal class we add restrictions on the **t_equal** property, limiting the range of this property to the same class as the domain. The **t_equal** property holds only between instances of the same class, in the Figure 5.11, this property fills the diagonal of the table. In the MADS-OWL, this property is defined as symmetrical and transitive. As an example, in Figure 5.13 we show the definition of the **t_equal** property for the **IntervalSet** temporal class.

The MADS constrained relationships are modeled in OWL by a predefined set of `owl:objectProperty` properties. As the name, constrained properties, suggests,

⁵the name is different due to the unique name assumption in Protégé.

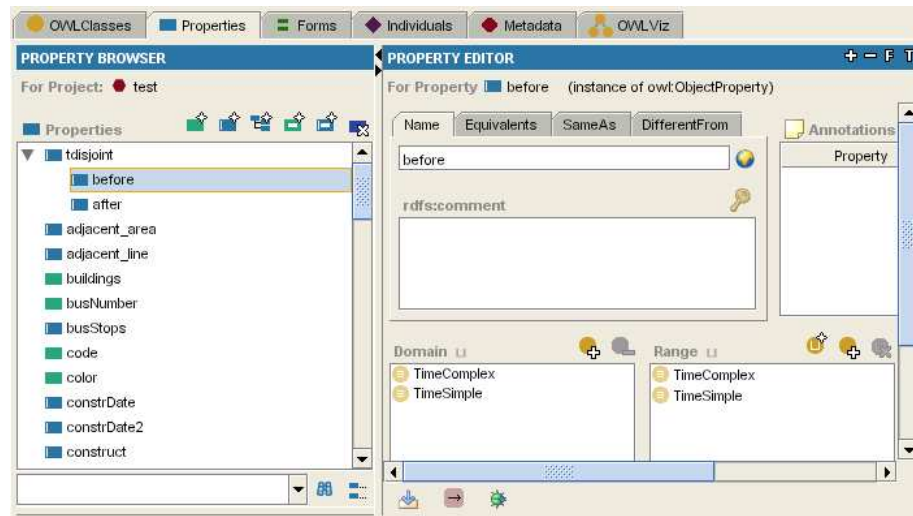


Figure 5.12: Disjoint synchronization relationship

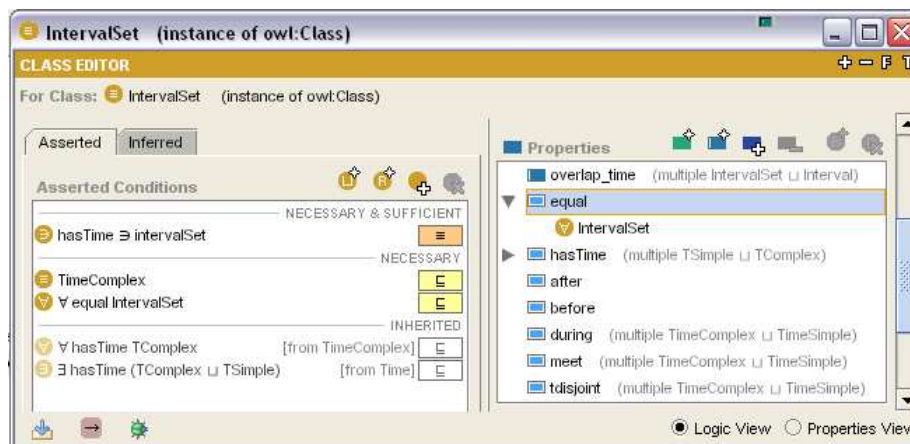


Figure 5.13: Temporal equal property.

for these special properties we have constrained their domains and ranges. This modeling choice has its disadvantages, but it is a viable way to implement this kind of MADS concepts. One of the goals in translating MADS schemas into OWL models, is to be able the use OWL reasoning services to uncover contradictory concepts in the original MADS schemas. For the constrained properties we cannot use reasoners to check the validity of domains and ranges of these properties. This is because the domains and ranges are not the constraints to be checked by the reasoner, they are used as axioms when reasoning. We avoid this fact in our modeling approach by limiting the choice of the model designer during the model definition phase. In other words, while creating a model, the designer cannot choose incompatible classes as domain or range for topological or synchronization properties. This goal is achieved due to the following modeling assumptions:

- The MADS model definitions in OWL, MADS-OWL, is presented as a reference ontology in each user-defined model. The MADS-OWL ontology is imported in the user model with a namespace `mads`. This namespace (or prefix) is used to distinguish the MADS concepts from the user model concepts. If the user wants to define a spatial or temporal class or property, he must use the MADS-OWL ontology classes and properties as superior for his/her model elements. User cannot change the imported ontology.
- By creating the user-defined properties as subproperties of MADS-OWL, the designer is limited in his choice of domain and range for these properties. As shown in Figure 5.12, subproperties inherit the domains and ranges of the superproperties. If the designer wants to rewrite the inherited values, his choice is checked for compatibility with parent values. If the designer defines a domain (or range) that is not a specialization of the domain (or range) of the parent property, his/her choice is rejected and an error message is displayed.

By using the approach described above, we insure that the designer will correctly define topological and synchronization properties in his/her model. The same approach as well limits the choice of instances that can participate in a constrained property. At instance creation phase, the designer is prompted to choose instances only from valid classes, i.e., temporal for synchronization, and spatial for topological. We summarize our modeling approach in the Table 5.4.

Table 5.4: Constrained relationships - MADS vs OWL

MADS constrained relationships	OWL constrained properties
Constrained relationships	Predefined set of OWL properties of type <code>owl:objectProperty</code>
Topological relationships	Topological properties with domain and range set to temporal classes
Synchronization relationships	Synchronization properties with domain and range set to temporal classes

5.2.5 Representation Dimension.

To model MADS representation dimension, we should provide the user with a way to add perception stamps to user models. There are two types of modeling primitives in OWL - classes and properties. As we described earlier, MADS object types are modeled by OWL classes, and MADS attributes and relationships are modeled by OWL properties (cf. Table 5.1). Classes and properties in OWL are two independent sets of modeling primitives, e.g., a property can be defined without any relationship

to any class. Contrary to the approach to define stamps in MADS, in OWL we define stamps separately for classes and for properties.

To provide for multiple perception in OWL, we add the `Perception` class to the definition of the MADS model in OWL. This `Perception` class is then extended by the user, i.e., the user defines stamps as subclasses of the `Perception` class along with a description for each stamp. The stamp descriptions are defined as OWL annotation properties of type `rdfs:comment`. The annotation properties can be added to OWL classes, properties and individuals. Since the user defines two different sets of perception stamps (one for classes, one for properties), the value of the annotation property can be the key to relate class and property stamps. We cannot use the name of the stamps for this purpose due to OWL unique name assumption. To associate a stamp to a user-defined class, it is sufficient to add the stamp as a superclass for the user-defined class. Besides the annotation, the stamps can have more properties. For example, if a stamp (as in [PSV05]) has two fields `viewpoint` and `resolution`, then two properties can be added to a user-defined stamp to model those two fields. By adding the stamp as a superclass for the user-defined class, the properties of the stamp are inherited in the user-defined class. Figure 5.14 shows an example with two perception stamps T_1 and T_2 and classes that inherit these perception stamps, i.e., belong to one or another perception.

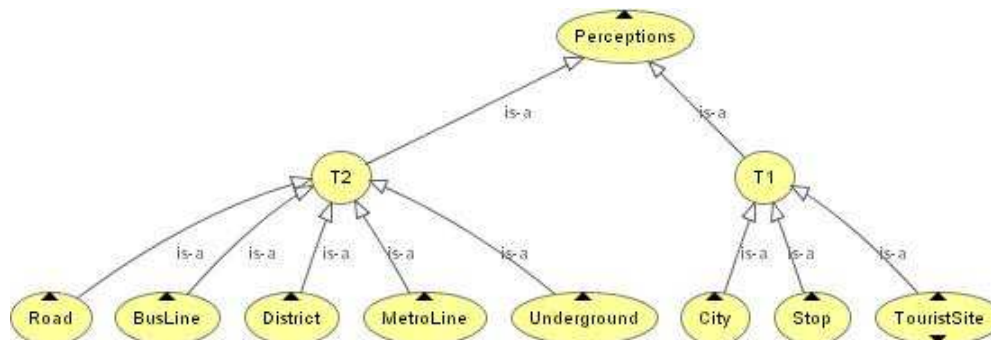


Figure 5.14: Perception stamps T_1 and T_2 with the corresponding classes

For the properties, we do similarly. First, we have defined the property `perceptions` that belongs to the MADS-OWL model. Then, the user can extend this property with user-defined stamps and descriptions. Figure 5.15 illustrates an example with two perception stamps t_1 and t_2 and different attributes and relationships that hold to these stamps. In the figure, `construct` and `seasons` are attributes and `busBy` is a relationship (cf. Figure 3.13 for MADS representation of the same elements). Also, we can see from the figure, that the attribute `season` holds both stamps as it appears as the subclass of both perceptions. With this approach for multiple perceptions we can translate MADS perception-varying object types in OWL, MADS inter-representation links, and MADS stamped relationships. The one type of the multiple representation that has no corresponding representation in OWL, is the stamps for `IsA` link. In other words, we cannot design an OWL

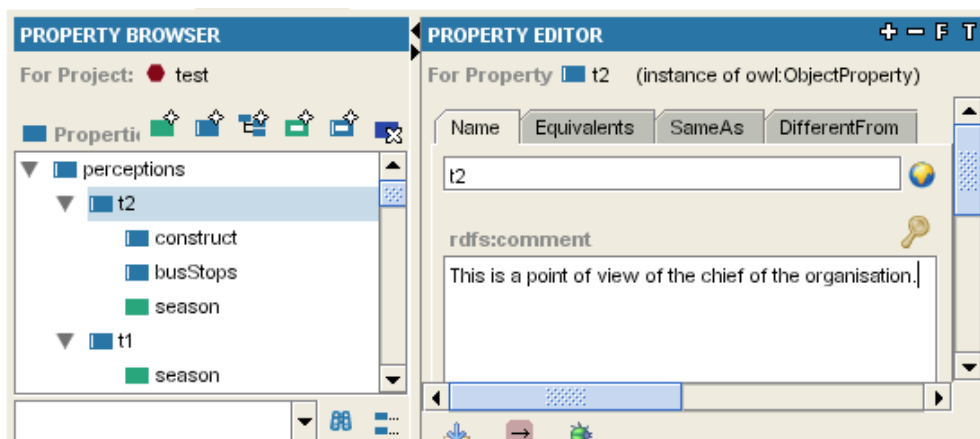


Figure 5.15: Perception stamps t_1 and t_2 with the corresponding classes

hierarchy of classes (properties) where the subclasses (sub-properties) inherit only a subset of the parent perception stamps. The definition of the OWL `owl:subClassOf` construct belongs to the OWL model and cannot be changed to meet the multiple perceptions paradigm. So, we limit the multiple perceptions mechanisms to the OWL model.

As we mentioned above, the two sets of OWL modeling primitives are independent and it is up to the designer to assign the perception stamps in a coherent way. It is up to the designer to check that the attributes of a class with the stamps T_1 and T_2 do not hold any other stamps than t_1 and/or t_2 . This ad-hoc perception assumption does not limit our approach to reasoning because the multiple perceptions of the MADS model are employed in structural patterns that are chosen after the reasoning is done. But, if needed, additional constraints can be added to the model to ensure the stamps coherence.

By this point we have described the translation of MADS structural, spatial, representation dimensions, along with constrained relationship to the OWL language. The resulting model, MADS-OWL is then used by the schema designer to create user models and state the inter-schema mappings for these models.

5.3 Schema Definition in OWL

With the MADS-OWL model, the designer can create his/her own models using the spatial, temporal, and representational dimensions of MADS-OWL. We detail the user schema definition in this section. For our reasoning services we employ a reasoner that does not provide for spatial or temporal reasonings. Thus, we defined the MADS-OWL model in a way it emulates reasoning over spatial and temporal elements of the user-defined model in the scope of the MADS data model.

The general idea on how to define a class as temporal/spatial is to make this class a subclass of one of the temporal/spatial classes predefined in MADS-OWL.

We illustrate the above modeling approach using our example schemas (Figures 3.12 and 3.13). Figure 5.16 depicts several classes from schema T_1 . As it is seen from the

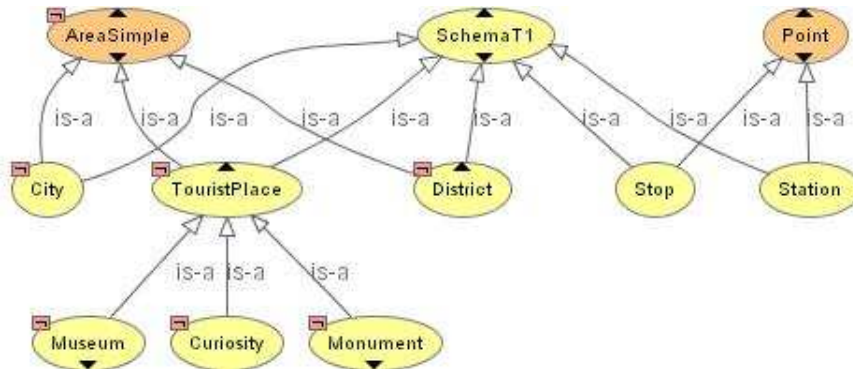


Figure 5.16: Some spatial classes from schema T_1 from Figure 3.13 defined in Protégé OWL.

figure, we have defined a class `SchemaT1` to group all the elements of the schema T_1 . Class `SchemaT1` is created purely for visualization reasons and it is not used for validation. Spatial object types `TouristPlace`, `City`, and `District` hold the **simple area** spatial semantics in the MADS schema thus, we defined them as subclasses of the `AreaSimple` spatial OWL class. The subclasses of `TouristPlace`: `Museum`, `Curiosity`, and `Monument` inherit spatial restrictions of their superclass. The `Stop` and `Station` spatial object types hold the **point** spatial semantics, thus in MADS-OWL they have the `Point` spatial OWL class as their superclass. As it is defined in MADS-OWL model, all the spatial classes are disjoint, i.e., a user-defined class cannot be a subclass of two spatial classes. Thus, as the figure shows, all the subclasses of `AreaSimple` are disjoint with subclasses of `Point`.

A spatio-temporal class is defined as a subclass of one of the temporal and one of the spatial MADS-OWL classes. As an example, in schema in Figure 3.12, there is a spatio-temporal object type `TouristPlace`. In OWL, we define this object type as a class `TouristPlace` with the `AreaSimple` spatial class and the `Interval` temporal class as its superclasses. Defined as such, the `TouristPlace` inherits spatial properties and restrictions as well as temporal.

To define a spatial attribute in MADS-OWL, we create an OWL property of type `<owl:objectProperty>` with the range restricted to a spatial class. For example, for the attribute `Start` of the object type `Walk` from Figure 3.13, we define the OWL property `start` with the domain restricted to the instances of the spatial class `Point`:

```
< owl:ObjectProperty rdf:ID = "start" >
  < rdfs:range rdf:resource = "Point" / >
  < rdfs:domain rdf:resource = "Walk" / >
< /owl:ObjectProperty >
```

Similarly, we define temporal attributes. In Figure 5.17 we show the definition of a complex temporal attribute `openTime` (cf. Figure 3.13 for MADS representation). The `openTime` attribute is a complex attribute with two sub-attributes `summer` and `winter`. To model such an attribute in OWL, we first create a class `MuseumOpenTime` that has two properties corresponding to two sub-attributes `summer` and `winter`. The

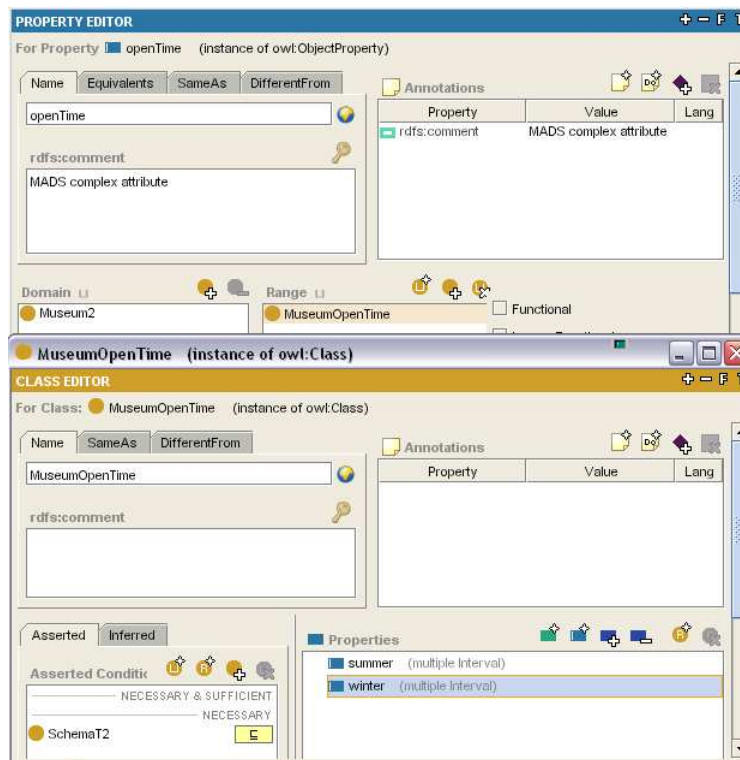


Figure 5.17: OpenTime attribute of the Museum object type from Figure 3.13 modeled in Protégé.

latter properties are temporal properties, i.e., the range of these properties is set to the temporal class `Interval`. This restriction guarantees that only instances of temporal class `Interval` can become values of the properties `summer` and `winter`. The `MuseumOpenTime` class is an auxiliary class that provides for the structure of the complex attribute `openTime`. Then, we define the property `openTime` and set its range to the `MuseumOpenTime` class. As we saw earlier, the ways MADS complex attributes and relationships are translated into OWL are exactly the same, thus the reverse process, i.e., translation from OWL to MADS is undefined. To avoid the ambiguity we have proposed to add a comment to the MADS complex attributes. For the `openTime` property we have added a `<rdfs:comment>` with the value "MADS complex attribute", Figure 5.17.

Properties in OWL are unidirectional, i.e., a MADS relationship is translated to OWL by two properties. Let us consider as example the `locatedIn` relationships be-

tween object types `TouristPlace` and `CityBorough` from Figure 3.12. This relationship holds topological inclusion semantics and therefore can link only spatial object types. In OWL, spatial property `locatedIn` is defined as a sub-property of the `include_area` topological property. This latter property belongs to the MADS-OWL model and restricts the domain and range of its sub-properties by those spatial classes that can participate in it, cf. table in Figure 5.9. The following OWL code describes the `locatedIn` property.

```
< owl : ObjectProperty rdf : ID = "locatedIn" >
  < rdfs : subPropertyOf >
    < owl : ObjectProperty rdf : ID = "include_area" / >
  < /rdfs : subPropertyOf >
  < rdfs : domain rdf : ID = "TouristPlace" / >
  < rdfs : range rdf : ID = "CityBorough" / >
  < owl : inverseOf >
    < owl : ObjectProperty rdf : ID = "locates" / >
  < /owl : inverseOf >
< /owl : ObjectProperty >
```

The range and domain of the `locatedIn` property are the subclasses of the range and domain of the `include_area` property. This restriction is verified by the Protégé ontology checker. The cardinality constraint for the `locatedIn` property is defined as a restriction in class `TouristPlace` on this property.

```
< owl : Classrdf : ID = "TouristPlace" >
  ...
  < rdfs : subclassOf >
    < owl : Restriction >
      < owl : minCardinalityrdf : datatype = "int" >
      > 1 < /owl : minCardinality >
      < owl : onProperty >
        < owl : ObjectPropertyrdf : ID = "locatedIn" / >
      < /owl : onProperty >
    < /owl : Restriction >
  < /rdfs : subclassOf >
< /owl : Class >
```

There is an inverse property `locates` that has `CityBorough` class as its domain and the `TouristPlace` class as its range. This property completes the definition of the MADS `locatedIn` relationship. In the class `CityBorough` we do not add any restriction on the property `locates` because by default, OWL properties are created as optional and multiple, which corresponds to the cardinality constraint `0:n`.

We have entirely defined schemas T_1 and T_2 (Figures 3.12 and 3.13) in OWL. Figures 5.18 and 5.19 show the hierarchy of the classes including spatial and temporal

super-classes. The properties are not displayed in the figures due to the complexity of the whole image.

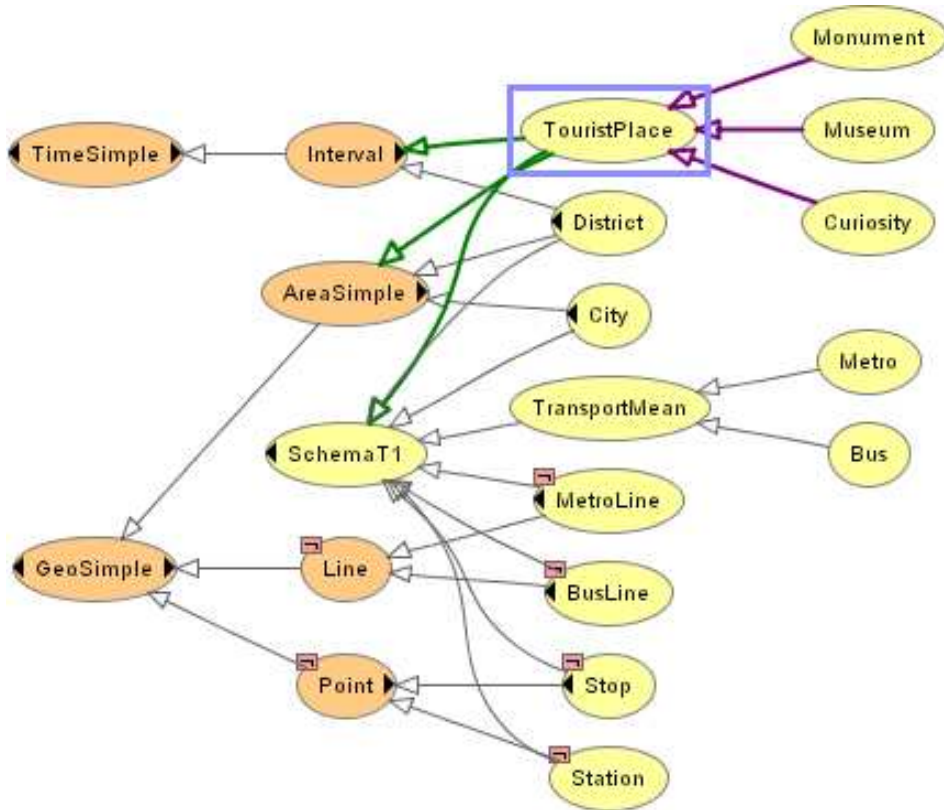


Figure 5.18: Schema T_1 in Protégé.

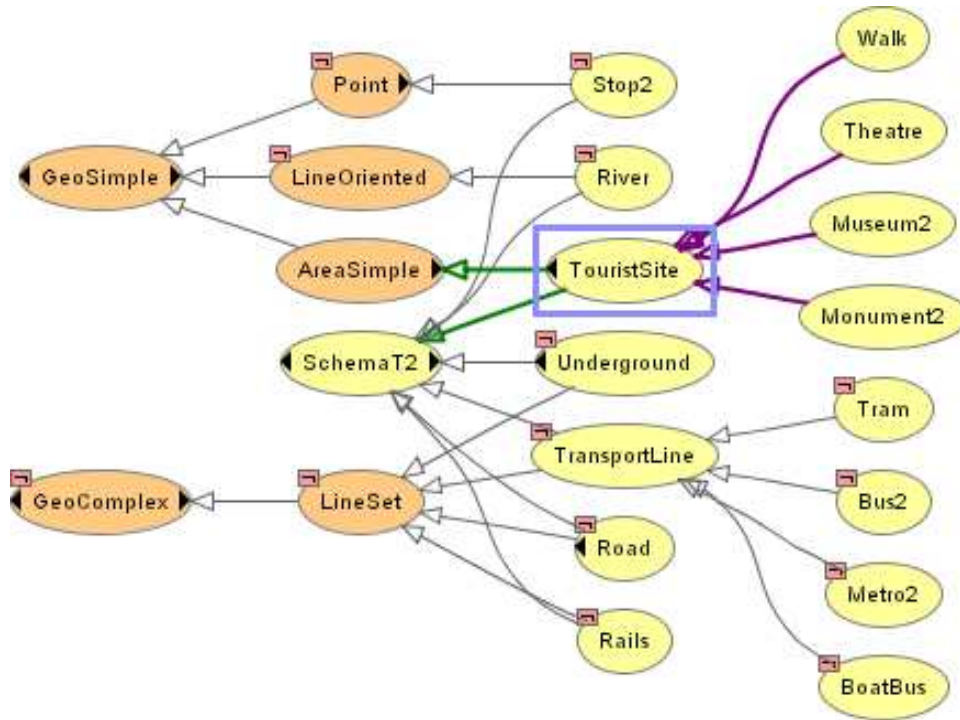


Figure 5.19: Schema T_2 in Protégé.

We now present the algorithm for MADS schema definition in OWL. In the notation used in the algorithm the subscript indicate to which model the element belongs, e.g., MADS object type A_{mads} has a corresponding OWL class A_{owl} .

ALGORITHM:

1. **OWL classes:**

- (a) Create an `<owl:Class rdf:ID "Aowl" />` for each object type A_{mads} in MADS diagram;
- (b) Create an `<owl:Class rdf:ID "Raowl" />` for each MADS relationship type Ra_{mads} with attributes;
- (c) Create an `<owl:Class rdf:ID "Rnowl" />` for each n -ary MADS relationship type Rn_{mads} ;
- (d) Create an `<owl:Class rdf:ID "Cowl" />` for each MADS complex attribute C_{mads} .

2. **OWL object properties:**

- (a) Create an

```

<owl:ObjectProperty rdf:ID "Rowl">
  <rdfs:domain rdf:resource "Aowl" />
  <rdfs:range rdf:resource "Bowl">
</owl:ObjectProperty>

```

for each binary MADS relationship type R_{mads} that links two MADS object types A_{mads} and B_{mads} ⁶;

- (b) Create an inverse object property

```
<owl:ObjectProperty rdf:ID "inverse_of_Rowl">
  <rdfs:domain rdf:resource "Bowl" />
  <rdfs:range rdf:resource "Aowl" />
</owl:ObjectProperty>
```

for each binary MADS relationship type R_{mads} that links two MADS object types A_{mads} and B_{mads} ;

- (c) Create an

```
<owl:ObjectProperty rdf:ID "Rcowl">
  <rdfs:domain rdf:resource "Aowl" />
  <rdfs:range rdf:resource "Cowl" />
  <rdfs:comment MADS complex attribute />
</owl:ObjectProperty>
```

for each MADS complex attribute c_{mads} , where A_{mads} is the object type the complex attribute c_{mads} belongs to;

- (d) Create n OWL object properties (with corresponding inverse object properties) for each n -ary MADS relationship Rn_{mads} (modeled as class Rn_{owl} in OWL) and for each MADS relationship with attributes Ra_{mads} (modeled as class Ra_{owl} in OWL);

3. OWL datatype properties:

- (a) Create a

```
<owl:DatatypeProperty rdf:ID "aowl">
  <rdfs:domain rdf:resource "Aowl" />
  <rdfs:range rdf:resource
    "http://www.w3.org/2001/XMLSchema type" />
</owl:DatatypePropertyProperty>
```

for each MADS simple attribute a_{mads} of a predefined data type of object type A_{mads} ;

- (b) Create a

```
<owl:DatatypeProperty rdf:ID "aowl">
  <rdfs:domain rdf:resource "Cowl" />
  <rdfs:range rdf:resource
    "http://www.w3.org/2001/XMLSchema type" />
</owl:DatatypePropertyProperty>
```

⁶ R_{owl} can later be defined as topological and/or synchronization property.

for each MADS composing attribute, i.e., for each of the composing attributes of a complex attribute c_{mads} .

4. MADS-OWL dimensions: structural, spatial, temporal:

(a) Create a

```
<rdfs:subClassOf>
  <owl:Class rdf:ID = "Aowl" />
</rdfs:subClassOf>
```

restriction for each MADS object type which is a subtype of A_{mads} ;

(b) Create a

```
<rdfs:subClassOf>
  <owl:Class rdf:ID = SADTowl />
</rdfs:subClassOf>
```

restriction for each A_{owl} class that models a spatial object type, where the S_{ADT}_{owl} is one of the predefined classes in the MADS-OWL spatial hierarchy;

(c) Create a

```
<rdfs:subPropertyOf>
  <owl:ObjectProperty rdf:ID = Topologicalowl />
</rdfs:subClassOf>
```

restriction for each R_{owl} object property that models a topological relationship type, where the $Topological_{owl}$ is one of the MADS-OWL predefined topological properties;

(d) Create a

```
<rdfs:subClassOf>
  <owl:Class rdf:ID = TADTowl />
</rdfs:subClassOf>
```

restriction for each A_{owl} class that models a temporal object type, where the T_{ADT}_{owl} is one of the predefined classes in the MADS-OWL temporal hierarchy;

(e) Create a

```
<rdfs:subPropertyOf>
  <owl:ObjectProperty rdf:ID = Synchronizationowl />
</rdfs:subClassOf>
```

restriction for each R_{owl} object property that models a synchronization relationship type, where the $Synchronization_{owl}$ is one of the MADS-OWL predefined synchronization properties;

5. Cardinality constraints, application specific constraints:

- (a) Add a

```
<owl:minCardinality
  rdf:datatype="http://www.w3.org/2001/XMLSchema int">
  n</owl:minCardinality>
<owl:maxCardinality
  rdf:datatype="http://www.w3.org/2001/XMLSchema int">
  m</owl:minCardinality>
```

for each of the MADS relationship types and attributes where the cardinalities are different from $0:n$;

- (b) Add different application specific restrictions, e.g., OWL necessary conditions, OWL necessary and sufficient conditions (to define classes), disjoint axiom on classes application constraints in the PAL language; specify OWL symmetric, functional, transitive properties;

END ALGORITHM

One of the assumptions that we make in our approach, is the validity of the source schemas. Thus at this stage of the translation from MADS to OWL, the ontology check for each schema should not give any error. The subsumption check should not produce any new subclass/superclass relationships since all the classes within each schema are mutually disjoint. Consistency check verifies if there are contradictory restrictions, for example, cardinality constraints on relationships (properties in OWL). In our example schema T_1 there is a relationship `stopServes` with the cardinality $1:n$, if the designer sets two cardinality restrictions that are incompatible, e.g., `<owl:minCardinality>` greater than `<owl:maxCardinality>`, then the class `Stop` which is the domain of this property, is found inconsistent. The checks provided by the RACER reasoner become more valuable when the inter-schema mappings are added.

5.4 Inter-schema Mappings Definition in OWL

Let us now consider the translation of MADS inter-schema mappings into OWL. There are three types of inter-schema mappings that are stated to relate source schemas. The first set of mappings is the *Schema population Correspondences* (SC) that relate the populations of object types. Then, there are *Property semantic Correspondences* (PC) for the attributes of the related object types. The last set of mappings is the *Matching Rules* (MR) which is a subset of the PC that relates identifier attributes. To illustrate the inter-schema mappings definition in OWL, we use the same set of MADS correspondences as we used for the theoretical part in Section 4.2.3, the syntax of MADS inters-schema mappings was presented in Section 3.4.2:

Schema population Correspondences:

- (1) $T_2.\text{TouristSite} \cap T_1.\text{TouristPlace}$;
- (2) $T_2.\text{Museum} \subseteq T_1.\text{Museum}$;
- (3) $T_2.\text{Monument} \subseteq T_1.\text{Monument}$;

Property semantic correspondences:

- (4) $T_2.\text{Museum} \bullet T_1.\text{Museum}$;
- (5) $T_2.\text{Monument} \bullet T_1.\text{Monument}$;
- (6) $T_2.\text{Monument}.\text{Construct} \equiv T_1.\text{Monument}$;
- (7) $T_2.\text{Museum}.\text{OpenTime} \equiv T_1.\text{Museum}$;
- (8) $T_1.\text{CityBorough}.\text{name} = T_2.\text{TouristSite}.\text{district}$;

Matching Rules within the Property semantic correspondences:

- (9) $T_1.\text{TouristPlace} \bullet T_2.\text{TouristSite}$;
- (10) $T_1.\text{TouristPlace}.\text{name} = T_2.\text{TouristSite}.\text{name}$;

Schema population Correspondences. In MADS, the *Schema population Correspondences* are stated between object types using the intersect, inclusion, equality, and disjoint operators - $\{\cap, \subseteq, \equiv, \emptyset\}$.

In OWL, the classes are assumed to overlap, i.e., if it is not explicitly forbidden by the disjoint axiom, OWL classes can have common instances, which corresponds to the overlapping relationship in MADS. For our OWL models **Schema1** and **Schema2**, we stated that within the schemas, the classes are disjoint, but classes belonging to different models do not have this restriction.

The inclusion in OWL is stated as the subclass axiom. In OWL, subclass means necessary implication, i.e., if the $T_2.\text{Museum}$ is stated as a subclass of $T_1.\text{Museum}$, then all the instances of the $T_2.\text{Museum}$ are the instances of $T_1.\text{Museum}$. This definition of subclasses in OWL corresponds exactly the inclusion relationship definition in MADS. For the population correspondences (1) - (3) we have stated the corresponding subclass axioms.

To state two classes as equivalent, it is sufficient to add a separate *necessary & sufficient* condition to one of the equivalent classes. The corresponding condition is added automatically to the other class. In the OWL syntax, the equivalence between $T_1.\text{Museum}$ and $T_2.\text{Museum}$ is stated as follows:

```
< owl : Class rdf : ID = "Museum" >
  < owl : equivalentClass >
    < owl : Class rdf : ID = "Museum2" / >
  < /owl : equivalentClass >
  ...
< /owl : Class >
```

Let us now run the reasoner with an invalid condition which we have asserted on

purpose. The condition we have added to the model is the following:

```
< owl : Class rdf : ID = "TouristPlace" >
  < owl : disjointWith >
    < owl : Class rdf : ID = "TouristSite" / >
  < /owl : disjointWith >
  ...
< /owl : Class >
```

This condition invalidates the previous one, because `TouristPlace` is the superclass of the `Museum`, and `TouristSite` is the superclass of `Museum2`, cf. Figures 5.18 and 5.19. The model asserts that with the disjoint super-classes some of their subclasses are equal, as shown graphically in Figure 5.20. If we check consistency for the `Museum`

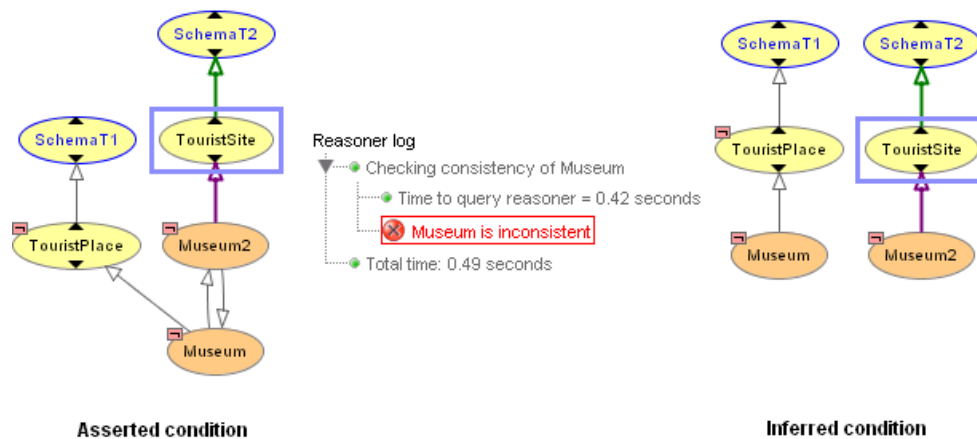


Figure 5.20: Validation for the `<owl:equivalentClass>` condition.

class, the reasoner finds it inconsistent. The inferred (consistent) model does not have the equivalence condition anymore.

Property semantic Correspondences. For the *Property semantic Correspondences* the set of operators includes the equality sign - \equiv - for attributes having comparable domain type like integer or strings; the set of topological relationships - \odot , \ominus , \oslash , \odot , \odot , \bullet - that are used to relate the spatial extensions of the schemas; and the set of synchronization relationships - \dashv , \dashv , \dashv , \dashv , \dashv , \dashv , \dashv - to relate the timestamped elements of the schemas.

The equivalence between two properties is stated with the `<owl:equivalentProperty>` restriction. For our OWL models, the equivalent properties are for example, the `nameCityBorough` and the `district` property of the `TouristSite` class (expression (8)) with the equivalence between them stated as follows:

```
< owl : DatatypeProperty rdf : ID = "district" >
```

```

...
< owl : equivalentProperty >
  < owl : FunctionalProperty rdf : ID = "nameCityBorough" / >
< /owl : equivalentProperty >
< /owl : DatatypeProperty >

```

The equality of the properties means that the ranges of these properties are the same.

For spatial mappings we use the MADS-OWL topological properties. For example, to state the expressions (4) in OWL, we add the `s_equal` property to the `Museum2` class with the closure axiom, i.e., with the existential and the universal restrictions. We add a similar restriction to the `Monument2` class to state the expression (5). By the definition of the `s_equal` property, its domain and range must be of the same type. If the designer states for example, that a museum is spatially equal to a bus stop, the reasoner will find the two classes inconsistent. To express synchronization mappings (6) and (7), we define a synchronization property `t_equal` between the `Construct` and the `Monument` classes and between the `MuseumOpenTime` and the `Museum` classes respectively.

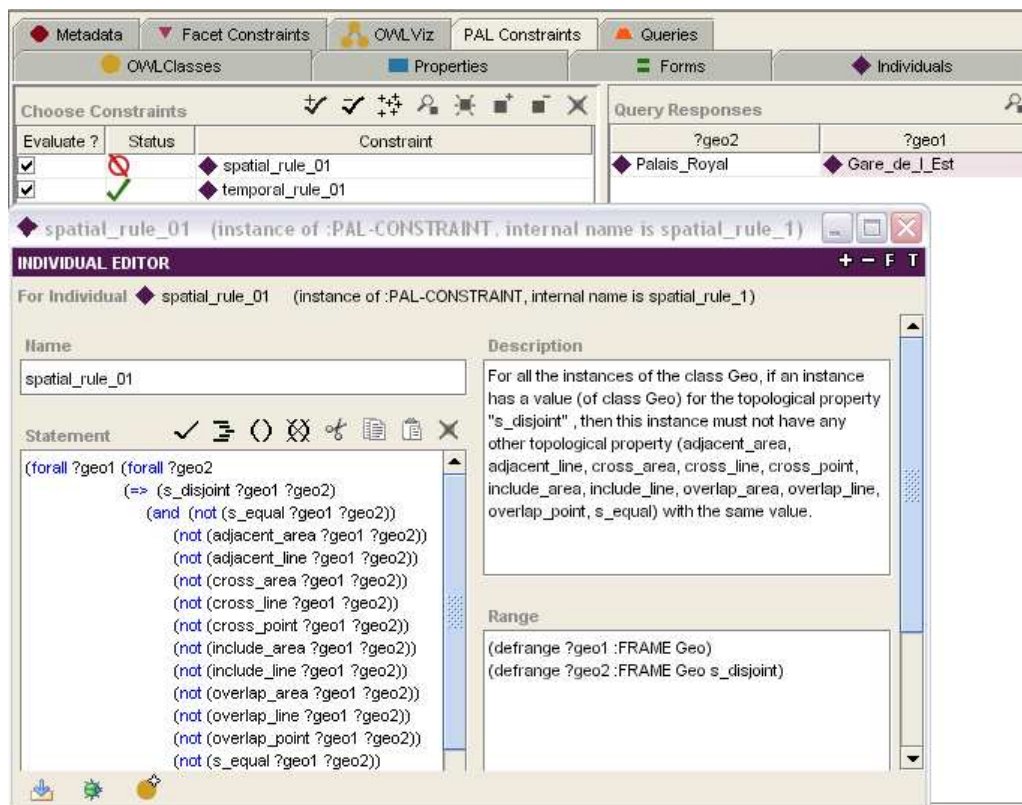


Figure 5.21: Constraint for the topological properties.

We have defined several constraints over the property semantic correspondences.

Since we are limited by the decidable subset of the first-order logic, we define the constraints on the instance level. In Figure 5.21 we show a constraint, `spatial_rule_01`, that prohibits an instance of the class `Geo` from having incompatible topological properties. To demonstrate the validation procedure, we have created two instances of the spatial class `Stop`, `Palais_Royal` and `Gare_de_L_Est`, and we stated in the model that the `Gare_de_L_Est` is spatially disjoint and spatially equal to the `Palais_Royal`. This last statement is inconsistent which had been found by the constraint validation engine. In Figure 5.21, the status of the constraint `spatial_rule_01` is *not valid*.

For the MADS-OWL synchronization properties we have defined a similar constraint, `temporal_rule_01`, that checks all the synchronization properties of a temporal instance. The constraint is invalidated if the instance has the same values for the `t_disjoint` and for any other of the synchronization properties. This type of constraints is equivalent to imposing the disjointness restriction on the `t_disjoint` synchronization property. But this constraint is a higher order constraint and would make our model undecidable in the frame of OWL-DL modeling approach. The constraint in the listing bellow is the `temporal_rule_01`, it is defined over the instances of the class `Time` and therefore is checked for any instance of any subclass of the class `Time`.

```
(defrange ?tmp1 :FRAME Time)
(defrange ?tmp2 :FRAME Time t_disjoint)

(forall ?tmp1 (forall ?tmp2
  (=> (t_disjoint ?tmp1 ?tmp2)
    (and (not (t_equal ?tmp1 ?tmp2))
      (not (overlap_time ?tmp1 ?tmp2))
      (not (meet ?tmp1 ?tmp2))
      (not (during ?tmp1 ?tmp2))
    )))
))
```

For the constraints we use PAL - Protégé Axiom Language that implements a model-checking rather than theorem-proving engine. In other words, PAL makes strong closed world assumptions and is used for writing restrictions on existing knowledge, not for asserting new knowledge. The primary goals of PAL are thus to detect incomplete entry of information and to check entered information for inconsistencies beyond the local scope of facets [PAL].

Now, considering the restrictions on the spatial and temporal MADS-OWL concepts, and additional PAL constraints over the spatial and temporal properties, we can emulate the spatial and temporal reasoning over the property semantic correspondences stated by the designer.

Matching Rules. As we described in Section 3.4.2, the identifier attributes can belong to one or more MADS dimensions, i.e., an object can be identified by its geometry, life-cycle, or a thematic attribute. The identifier attributes are stated by

the schema designer. As in databases, the identifier properties in OWL are mandatory, adding to this fact the unique name assumption, we can use a restriction over an identifier attribute as *necessary & sufficient* condition that defines the class that holds this property. The properties that figure in the *necessary & sufficient* conditions are the candidates for the matching rules. In our example, the properties that identify the tourist attractions are the name and geometry associated to the instances of the classes `TouristSite` and `TouristPlace`, i.e., instances of these classes that have the same values for the `name` and `hasGeometry` properties, are the same instances. Mapping (9) in OWL, is stated with the `<owl:equivalentProperty>` restriction:

```
< owl : DatatypeProperty rdf : ID = "nameTouristPlace" >
...
  < owl : equivalentProperty >
    < owl : FunctionalProperty rdf : ID = "nameTouristSite" / >
  < /owl : equivalentProperty >
< /owl : DatatypeProperty >
```

Mapping (10) is stated as the restriction over the `s_equal` property in the class `TouristSite` as shown:

```
< owl : Class rdf : ID = "TouristSite" >
  < rdfs : subclassOf >
    < owl : Restriction >
      < owl : allValuesFrom rdf : resource = "TouristPlace" / >
      < owl : onProperty >
        < owl : ObjectProperty rdf : ID = "s_equal" / >
      < /owl : onProperty >
    < /owl : Restriction >
  < /rdfs : subclassOf >
  ...
< /owl : Class >
```

The names and the geometries of the tourist attraction instances are their identifying attributes. We add the existential restrictions in the *necessary & sufficient* set of restrictions thereby defining the `TouristSite` and the `TouristPlace` classes. If the models `Schema1` and `Schema2` are populated with the instances of the above mentioned classes, the reasoner will automatically infer the identical instances based on their name and geometry values.

To summarize the representation of the MADS inter-schema mappings in MADS-OWL we now make a list of the translations for each MADS inter-schema mapping operator, we assume that the inter-schema are stated between two MADS object types A_{mads} and B_{mads} with corresponding MADS-OWL classes A_{owl} and B_{owl} .

TRANSLATION:

1. **Schema population Correspondences (SC).** Add to the definition of the class A_{owl} one of the following MADS-OWL constructors:

<u>MADS</u> SC operators	<u>MADS-OWL</u> representation
\cap	<owl:DisjointWith/> for all non-overlapping classes;
\subseteq	<owl:subClassOf/>;
\emptyset	<owl:DisjointWith/>;
\equiv	<owl:sameClassAs/>;

2. **Property semantic Correspondences (PC).** Add appropriate restrictions to the definition of the A_{owl} class:

<u>MADS</u> PC operators	<u>MADS-OWL</u> representation
=	<owl:equivalentProperty/>;
\leftrightarrow	<owl:equivalentProperty/>;
any topological operator, i.e., $\bigcirc \bullet$, $\bigcirc \bullet$, \mathcal{D} , \odot , \odot , \bullet	<owl:Restriction> <owl:onProperty> <owl:ObjectProperty rdf:about= <i>Topological_{owl}</i> /> </owl:onProperty> <owl:allValuesFrom> <owl:Class rdf:ID= B_{owl} /> </owl:allValuesFrom> </owl:Restriction>
any synchronization operator, i.e., \dashv , \equiv , \dashv , \dashv , \dashv , \dashv , \dashv	<owl:Restriction> <owl:onProperty> <owl:ObjectProperty rdf:about= <i>Synchronization_{owl}</i> /> <owl:onProperty> <owl:allValuesFrom> <owl:Class rdf:ID= B_{owl} /> </owl:allValuesFrom> </owl:Restriction>

3. **Matching Rules (MR).** The PCs with an equivalence operator, including spatial (\bullet) and temporal (\equiv) equivalence, are the matching rules.

5.5 Chapter Summary

In this chapter, we described a hybrid approach exploiting advantages of two formalisms: a spatio-temporal conceptual model and an expressive description logic. The two approaches we employ come from different worlds: database and ontology. *"In the former, validity is possible but doubt is a plague, in the latter, the doubt is embraced, but validity is a myth"* [HPPSH05]. We do not enhance any of the two approaches to achieve our goal - meaningful integration of spatio-temporal database schemas. Instead, we exploit the advantages of each approach and cope with their limitations. From the database world, we use MADS, an EER conceptual model, intended to describe spatio-temporal application data. A peculiar feature of MADS that is of interest in an integration environment is that it includes specific concepts to describe multiple representations of data. Indeed, as stated in [DPS98], full integration of spatial database requires a powerful data model for the integrated schema in order not to lose the semantics of the original schemas. EER like conceptual schemas are concise if compared to the description logics ones; they represent the real world in the way humans think of it: by objects with their properties and relationships. With the strong representation capabilities, conceptual models lack a validation support that becomes a crucial feature for schema integration processes. As shown in Figure 4.1, our integration process requires additional information to be stated by the designer. Initially, integrity constraints and inter-schema mappings are not part of the source schemas; addition of these statements may entail inconsistencies in the resulting description. To validate user input, the source schemas together with the integrity constraints and the inter-schema mappings are translated into a description logic language. The translated model is then checked for satisfiability. In the context of our integration process, the satisfiability of the model means that the inter-schema mappings conform to the data model, that they are mutually non-contradictory, and the integrity constraints imposed on the data model and the user models are compatible. Furthermore, the successful satisfiability check allows the schema designer to follow any of the integration policies, i.e., minimal, non-exhaustive, maximal, or preservation (Section 3.4). In the case of a non-satisfiable model, only the preservation policy with the multiple representation structural patterns is a viable solution for the integrated schema.

We have defined the MADS-OWL model which is the MADS translation into the OWL language. MADS structural, spatial, temporal, and representation dimensions are orthogonally defined in the OWL ontology language. We were not able to translate all the MADS features into OWL. In the structural dimension we could not for example define relationships with attributes. The solution for this shortcoming of the OWL semantics is to remodel a relationship with attributes by an object type with attributes, as shown in Figure 5.4. Using the binary structures OWL we had to use same constructors for complex attributes and for relationships, adding an annotation property to distinguish them. For the spatial and temporal dimensions we were not able to implement the overloading concept along the spatial

types hierarchy. For example for the schema in Figure 3.13, the `Walk` object type in MADS-OWL model holds the same spatial semantics as its parent, i.e., the `simple area` type. For MADS representation dimension, we implemented the perception stamps in different and independent ways for classes and properties. This modeling solution requires additional constraints to be added to the model to link two sets of stamps. Due to the OWL definition of the subclass constructor, we could not associate perception stamps to the `IsA` (`subClass` constructor in OWL) links. This does not limit the validation process since we do not reason over the models with different perception stamps; perceptions stamps may be associated to the integrated schema in MADS provided the designer chooses the preservation integration policy.

The MADS-OWL model is imported as a reference ontology in the user model. Then, the user defines spatial or temporal classes or properties by using the MADS-OWL ontology classes and properties as parent for his/her model elements. If the designer wants to rewrite the inherited values, his choice is checked for compatibility with parent values. If the designer defines a domain (or range) that is not a specialization of the domain (or range) of the parent property, this choice is rejected and an error message is displayed. The integrity constraints associated to the data model are inherited by the user models and can be validated in our modeling framework. Figure 5.21 shows a screen-shot with an invalidated constraint `spatial_rule_01` and spatial instances `Palais_Royal` and `Gare_de_L_Est` that do not conform to this constraint.

Our modeling approach insures that the model designer will correctly define spatial and temporal elements as well as the topological and synchronization properties. A model constructed with the reference MADS-OWL model, is then checked for satisfiability considering MADS spatial and temporal semantics.

Chapter 6

System Modeling Issues

This chapter is devoted to the system modeling. We present two models, UML and SEAM model, of a system that provides for integration functionalities. The models describe a framework where several tools are employed together, each involved in the service it is best suited for. Unified Modeling Language (UML) is a non-proprietary, third generation modeling and specification language. However, the use of UML is not restricted to modeling software. As a graphical notation, UML can be used for modeling hardware (engineering systems) and is commonly used for business process modeling, representing organizational structure, and systems engineering modeling (from *wikipedia.org*). SEAM [Weg03] stands for Systemic Enterprise Architecture Methodology. The core idea of SEAM is to provide a uniform notation for all enterprise stake-holders that participate in a business process. SEAM enterprise model supports hierarchical modeling approach, where a sub-level model details its super-level model using a uniform notation. In SEAM, at each level, a system can be represented with a computational viewpoint (i.e., as a collaboration of subsystems) and with an informational viewpoint (i.e., focusing on the semantics of the information and information processing performed) [BRW03].

6.1 UML modeling

In this section we present several UML diagrams for integration tool - *ICATool*. The tool allows to read MADS schemas and verify the syntax of the *Inters-schema Correspondence Assertions* or inter-schema mappings between these schemas. The modeling approach is adopted from [RS01]. In the following sections we present the use case, the robustness diagram, and the activity diagram for ICATool.

6.1.1 Use Case introduction and definitions

Use case modeling is a powerful, industry standard requirements modeling technique. The use case model is depicted in diagrams using UML. Use cases are used to model how the system will interact with users and external systems. Use cases provide

an important base upon which the system can be designed and built. The basic definitions of the use case vocabulary [Coc00]:

- *Actor* - anyone or anything with behavior;
- *Stakeholder* - someone or something with a vested interest in the behavior of the system under discussion (SuD);
- *Primary actor* - the stakeholder who or which initiates the interaction with the SuD to achieve a goal;
- *Use case* - a contract for the behavior of the SuD;
- *Scope* - identifies the system that we are discussing;
- *Preconditions and guarantees* - what must be true before and after the use case runs;
- *Main success scenario* - a case in which nothing goes wrong;
- *Extensions* - what can happen differently during the scenario.

A use case captures a contract between the stake-holders of a system about its behavior. The use case describes the functionality of the system - what the system will do for the user in order to get some useful work done. It also helps to layout the actors or users of the system and their role in running the system. The use case describes the system's behavior under various conditions as the system responds to a request from one of the stake-holders, called the *primary actor*. The primary actor initiates the interaction with the system to accomplish some goal.

A use case description generally includes:

1. General comments and notes describing the use case;
2. *Requirements* - things that the use case must allow the user to do, such as ability to update order, ability to modify order etc.
3. *Constraints* - rules about what can and can not be done. The rules include pre-conditions that must be true before the use case is run - e.g. create order must precede modify order; also include post-conditions that must be true once the use case is run e.g. order is modified and consistent; invariants - these are always true - e.g. an order must always have a customer number
4. *Scenarios* - sequential descriptions of the steps taken to carry out the use case. May include multiple scenarios, to cater for exceptional circumstances and alternate processing paths;
5. *Scenario diagrams* - sequence diagrams to depict the workflow.

In the following we propose a use case for the *ICATool*, the tool that allows for creating the inter-schema mappings for MADS schemas. As the input, the editor takes two or more MADS schemas and following the integration method (Section 3.4) proposes an integrated view of the input schemas.

6.1.2 Use Case: ICATool

Use Case : Generate the Integrated Schema.

Primary Actor: Database Integrator (DI).

Scope: ICATool integration tool.

Stake-holders and Interests:

Domain Expert (DE) - assists the DI to obtain adequate results.

ICATool - based upon the correct inter-schema mappings generate the correct integrated schema.

MADS schema editor - displays the schemas (input and integrated) in graphical form for DI and DE to analyze.

Description Logic Reasoner (DLR) - validates MADS schemas and inter-schema mappings, based on the validation results, DI chooses the integration policy for the final schema.

Preconditions: the DI is knowledgeable in the domain of conceptual modeling and has at least theoretical knowledge about the integration method used in the ICATool; there is an expert support accessible to the DI; the schemas that are used as the input of the ICATool are in MADS format; these schemas are valid.

Minimal Guarantees: the DI is presented the local (input) schemas in the textual format and gets some knowledge of the data modeled by these schemas.

Success Guarantees: the DI obtains the integrated schema that meets the requirements of the application this schema is generated for.

Main Success Scenario:

1. DI starts the ICATool.
2. DI chooses the MADS schemas.
3. The schemas are parsed and displayed by the ICATool.
4. DI analyzes the schemas and states the inter-schema mappings.
5. DI starts the syntactic parser of the ICATool.

6. DI interprets the result of the parsing, makes corrections if necessary.
7. The inter-schema mappings and schemas are translated to MADS-OWL language and sent to the DLR for validation.
8. DI chooses the integration policy to be followed.
9. DI starts the integrated schema generator function of the ICATool.
10. The integrated schema is displayed in the ICATool window in the textual form and in the MADS editor window in the graphical form.
11. DI analyzes the result schema and either accepts or refuses it.

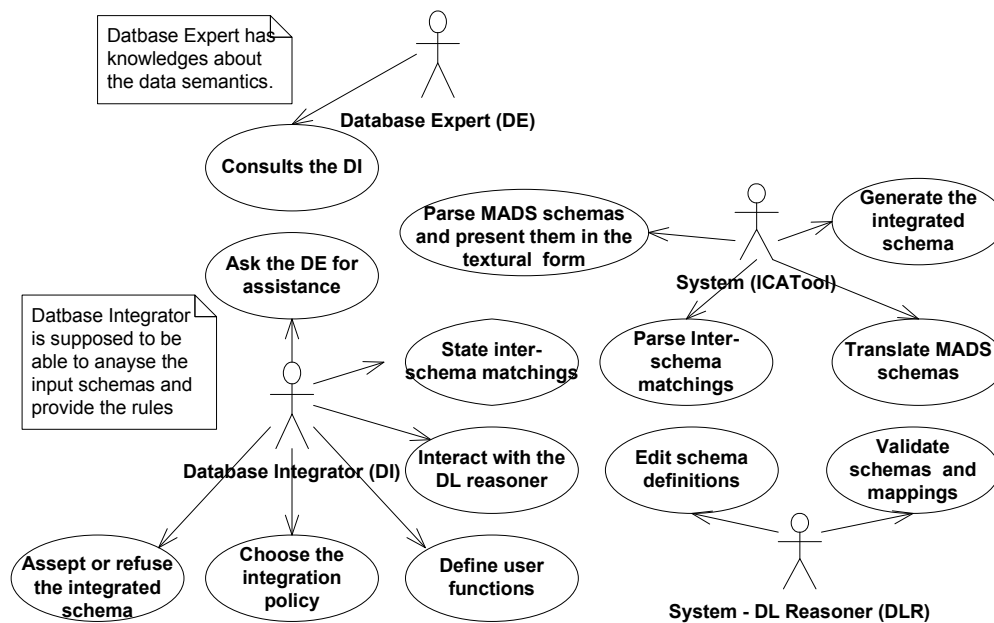


Figure 6.1: The Use Case for the ICA Tool.

Extensions:

- 4a. DI has insufficient knowledge to formulate the inter-schema mappings.
 - 4a.1. DI consults the DE, together they formulate the inter-schema mappings.
- 6a. The parser results in syntactical mistakes.
 - 6a.1. DI asks the DE for assistance.
- 11a. DI needs help in interpreting the results of integration.
 - 11a.1. DI asks the DE for assistance.

In the following two sections we present two modeling diagrams - robustness and activity diagram. These two diagrams are similar in the sense that they model the general interaction flow between stake-holders of the system, and give a first view on the functions to be implemented in the application. These diagrams are transitional from the static “what to model” to dynamic “how to” implement the system. We aim, by presenting hereafter these two diagram types to compare their expressiveness and usefulness.

6.1.3 Robustness diagram for the ICATool

In the UML system design, how to get from use cases to sequence diagram is a non-trivial problem. Most approaches that can be found in the literature, talk about use cases and sequence diagrams but do not address how to get across the gap between the fuzzy use cases and a code-like level of detail on the sequence diagrams [RS01]. To close this gap the [RS01] proposes a diagram called *robustness diagram*. The place of the robustness analysis is between what the system has to do and how it is actually going to accomplish this task. A robustness diagram is similar to the UML activity diagram, in that it shows the objects that participate in the scenario and how these objects interact with each other. In a robustness diagram the following three elements are used (see Figure 6.2):

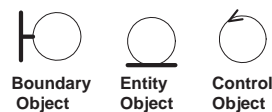


Figure 6.2: Robustness Diagram Elements.

- **Boundary object** - used by actors use to communicate with the system. Usually, boundary objects include screens, dialogs, and menus;
- **Entity objects** - usually objects from the domain model. Entity objects map to the database tables and files that hold the information;
- **Control objects** - embody much of the application logic. They serve as connectors between the users and the stored data.

There are four basic rules that a robustness diagram should meet (see Figure 6.3):

1. Actors can only talk to the boundary objects;
2. Boundary object can only talk to controllers and actors;
3. Entity objects can only talk to the controllers;

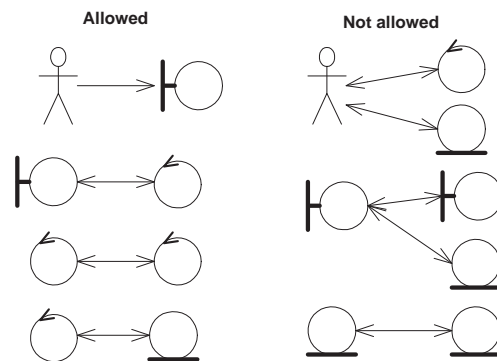


Figure 6.3: Robustness Diagram Rules.

4. Controllers can talk to boundary objects, entity objects and other controllers, but not to actors.

Following the use case that was presented earlier Section 6.1.2, we propose the robustness diagram for this use case (see Figure 6.4). This robustness diagram follows the course of actions included in the use case.

As we mentioned before, the robustness diagram provides a bridge between the “analysis level” view provided by the text of the use case and the “detailed design” view that would be presented on a sequence diagram. Since it’s very difficult to proceed from analysis directly to detailed design, it’s hard to do modeling successfully without this step. In the following section we present the activity diagram for the same use case.

6.1.4 Activity diagram for the ICATool

UML activity diagrams document the logic of a single operation or method, a single use case, or the flow of logic of a business process. In our case we will design an activity diagram for the ICATool (Inter-schema Correspondence Assertions Tool) use case. To give a broader overview of the activity diagram design and usage we begin with a general description of this type of diagram. To create a UML activity diagram, the following steps should be iteratively performed.

1. *Identify the scope of the activity diagram.* The scope could be a single use case, a portion of a use case, a business process that includes several use cases, or a single method of a class. Once the scope is identified, a note indicating an appropriate title for the diagram should be added to it.
2. *Add start and end points.* Every activity diagram has a starting point and an ending point. Some authors make the ending points optional ([RS01]). Sometimes an activity is simply a dead end, but even in this case, indicating the only transition to an ending point would not introduce any overwhelming

For the diagram depicted in Figure 6.5, the scope of the diagram is integrated schema generation with the ICATool. This process can be finished either with success - the schema is generated and it meets the application requirements; or with failure, but still fulfilling the minimal guarantees (see the use case in Section 6.1.2). The failure case can occur if the database integrator has insufficient knowledge either on the integration method, or on the semantics of the modeled data and application. The set and the order of the activities correspond to the lines in the use case.

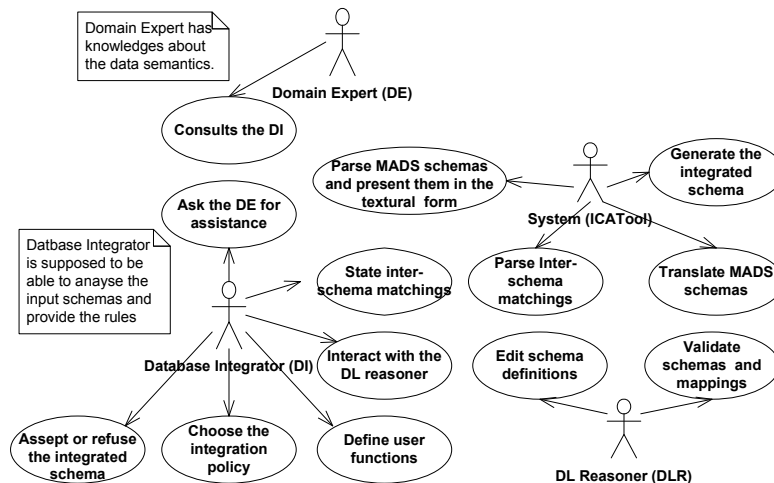


Figure 6.5: Activity Diagram for the ICATool.

Discussion. In this section we make a comparison of the two diagram types that were presented in the Sections 6.1.3 and 6.1.4. These diagrams generally model the interaction flow in the system. The elements representing the same logic in both diagrams are the *activities* in the Activity diagram (AD) and *controllers* in the Robustness diagram (RD). From the point of view of the functionality of the future application, an RD draws a more precise picture of the potential functions to implement. The activities in an AD are more general than the controllers in an RD. The controllers of an RD diagrams can be attributed not only to actors, as in an AD, but also to the elements that constitute these actors. For example, if one of the actors is a tool, then a controller can represent one of the functions of this tool. On the other hand, analyzing an RD, it is difficult to say which actor initiated what function. A RD is written for every use case, whereas a single application can be specified by several use cases that intersect. In an AD it is possible to specify the actor that initiates an activity.

Structurally, an AD is richer than an RD. The significant elements of an AD such as termination points, decision points, parallel execution are not easily perceived from an RD. The termination points allow to define precisely the possible final states of the modeled process and link them to the following process flow. For a coherent system design it is important to clearly see the possible outcomes of the

use cases composing the application. On the other hand, an RD displays *entity* concepts which are the data sources that are involved in the process flow. Such an approach to modeling the data sources in an RD diagram, is a simple and condense way to model the process flow with the underlying data. There in no other diagram in UML that would combine the static and dynamic elements of the application. To conclude our comparison, we would like note that there are rules for an RD design that check the application logic, whereas an AD is designed without ordering constraints. This could defer logic mistakes discovery for further modeling phases where correction of such mistakes becomes more expensive.

6.2 SEAM modeling

In the previous section we have presented the use case, robustness and activity UML diagrams for ICATool, a tool to design an integrated MADS schema. Although all these diagrams describe the same artefact at different levels, they use different graphical notations with hardly traceable links between them. It seem beneficial to employ a model that provides for a unified notation in processes and resources modeling. The state of the art in hierarchical object-oriented models is such that no models are suited for that purpose [LW04]. Existing hierarchical models either represent the hierarchy in an inconsistent way, or focus on only one system of interest, or inadequately model actions, or lack a complete metamodel to keep the traceability across the hierarchy. Authors in [LW04] present a meta-model SEAM, that defines a hierarchical object-oriented model of systems such as those found in enterprise architecture. SEAM stands for the Systemic Enterprise Architecture Methodology, described in [Weg03]. In general, SEAM distinguishes 3 organizational levels: business, operation and IT (Information Technology) [Weg03]. The first level represent companies working together. The second level represents people and systems within the company. The third levels represents how the IT system is built. However, many other levels can exist (depending on the particular organization and modeler).

Let us imagine that ICATool is employed in an organization that implements the mediation level services for organizations like the one we described in the very beginning of this thesis (Figure 1.1). For that organization we model only two SEAM levels, i.e., business and operation. The IT level, which is out of the scope of this thesis, usually represents the system of interest as a collaboration of components - IT systems - supporting the process defined on the operation level. Using SEAM approach to model the Enterprise Architecture (EA) of such an organization, the Project Manager (PM) would first design the model of the Business Organizational Level (Figure 6.6). A business level model shows the system of interest within its environment and defines the collaboration(s) between main participants. On this modeling level, the PM defines the process to implement (i.e., Integrated Schema Design), the actors of the process (i.e., Domain Expert (DE), Integrated Schema Designer (ISD), ICATool, and Description Logic Reasoner (DLR)) and their main

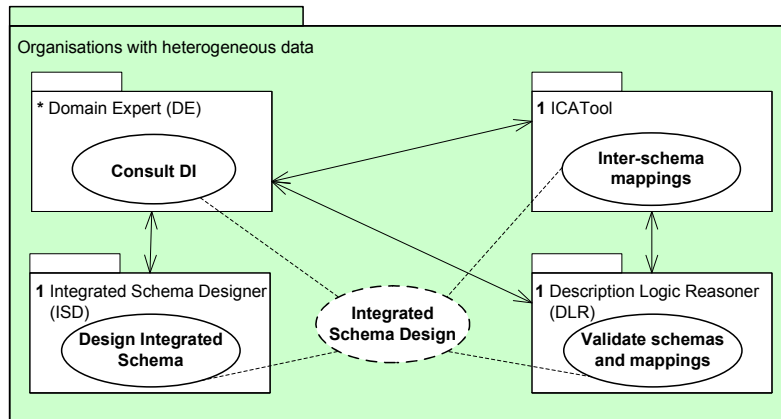


Figure 6.6: Business Level Model

functionalities, as shown in Figure 6.6.

Further detailing the *integrated schema design* process, the process manager designs the Operation Organizational Level models. The operation level considers the process (e.g., a business process) within the system of interests. Such process describes the role (the functionality) of the system in the collaboration defined on the business level. In Figure 6.7, we show the organizational level diagram for one of the actors, i.e., Integrated Schema Designer. Although the model is designed for a single

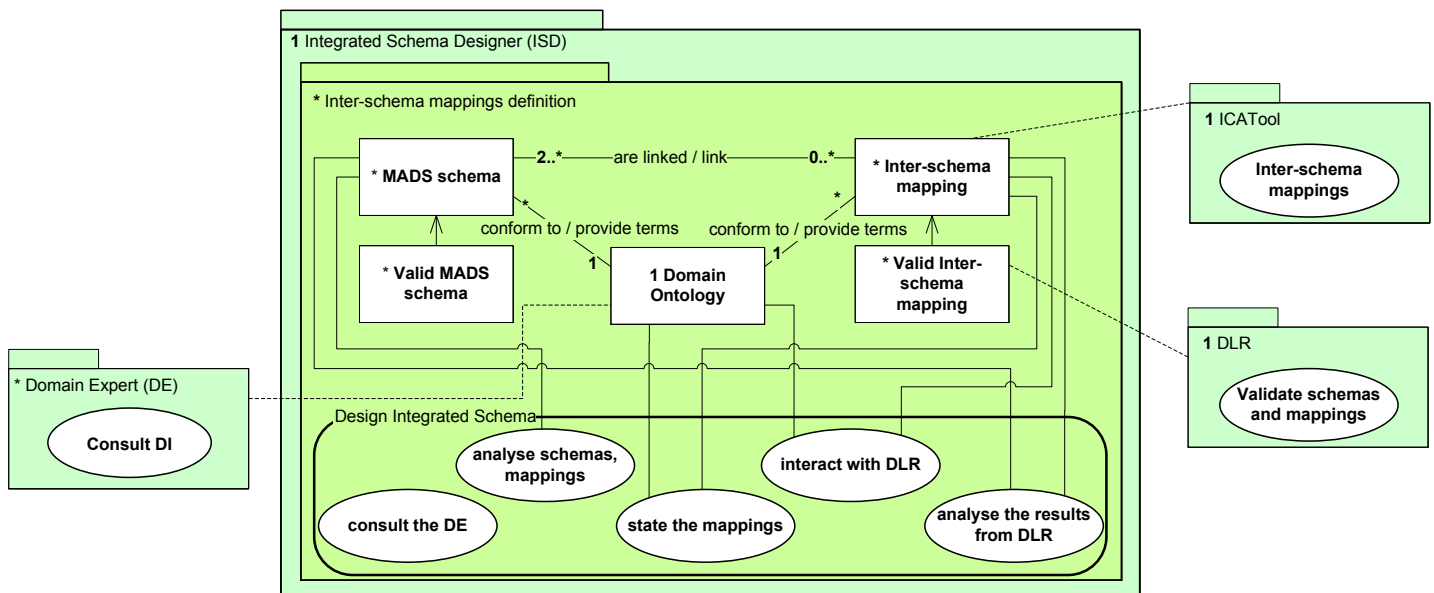


Figure 6.7: Operation Level Model

participant, it preserves the relationships between all participants in the process and adds links from all the participants to the data they use. Thus, the operation level model shows the object model, i.e., object types that are used to accomplish the

process, and the detailed functionality of one participant. This model integrates the UML object model with a detailed use case. The advantage in such a representation, is that the object model is explicitly presented (contrary to the UML diagrams that are different for data and processes). Also, is it clear from the model, what are the component processes of the main functionality of the participant modeled. This SEAM operation level model could be improved by modeling the *integrated schema design* process in an explicit way.

For that type of modeling, the Business Process Modeling Notation (BPMN, www.bpmn.org) could be considered. BPMN is designed to cover many types of modeling and allows the creation of process segments as well as end-to-end business processes, at different levels of precision. The primary goal of the BPMN is to provide a notation that is readily understandable by all business users, from the business analysts that create the initial drafts of the processes, to the technical developers

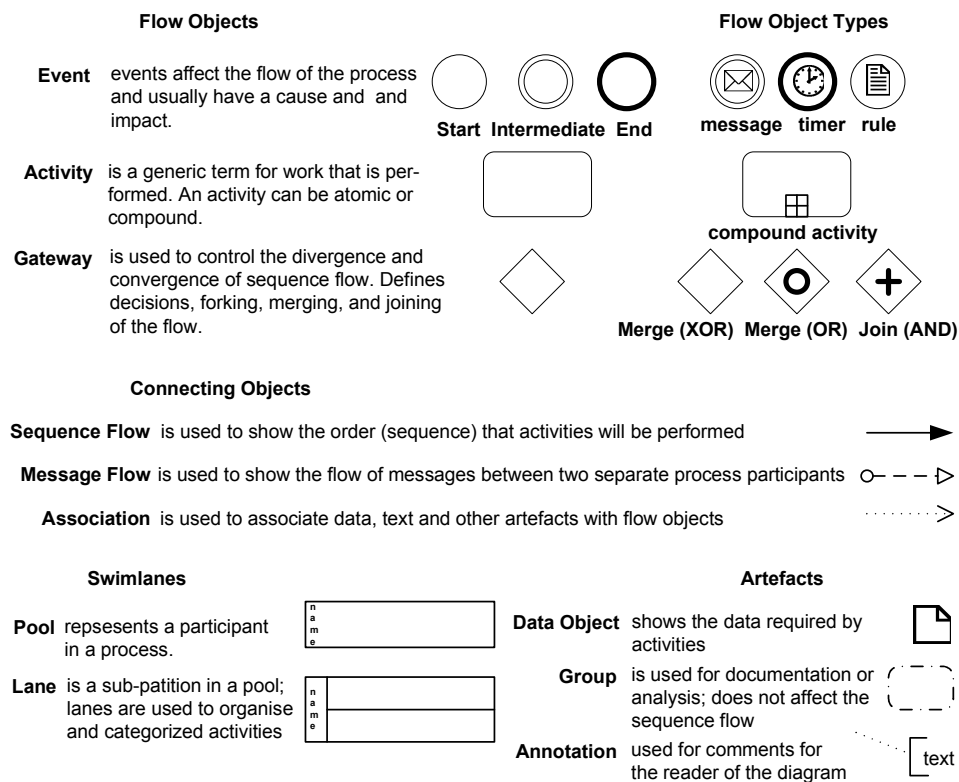


Figure 6.8: BPMN elements.

responsible for implementing the technology that will perform those processes, and finally, to the business people who will manage and monitor those processes [Whi04]. BPMN defines a Business Process Diagram (BPD), which is based on the flowcharting technique tailored for creating graphical models of business process operations. A BPD then, is a network of graphical objects, which are activities and the flow controls that define their order of performance [Whi04]. One of the drives for the development of BPMN is to create a simple mechanism for creating business process

models, while at the same time being able to handle the complexity inherent to business processes. The four basic categories of elements are: *Flow Objects*, *Connecting Objects*, *Swimlanes*, and *Artifacts*, Figure 6.8 shows basic elements provided by the BPMN.

For our *integrated schema design* process, we define one pool (i.e., one process) with 3 departments (i.e., sub-processes). Each of the sub-processes contains the activities of one tool employed by the Integrated Schema Designer to construct an integrated schema (Figure 6.9). One of the advantages of the BPMN, is that the reader of the diagram sees the process as the whole, and at the same time the activities of each component sub-process are clearly separated. In the BPD in Figure 6.9, we used the events (i.e., start event and end event), gateways, activities, associations, and data objects. The BPD is close to the activity diagram from Figure 6.5; the process we model is a simple one, so we cannot demonstrate all the advantages of an BPD over an AD, but it is clear that a BPD is more expressive and rich than an AD. To benefit from the two expressive notations, we believe that a combination of the SEAM operation level notation and the BPMN¹, to have in one diagram a detailed process model and corresponding class model, would not compromise the readability of an operation organizational level model.

¹Currently such a synergy of notations is under development in LAMS laboratory and the draft name of this method is S-BPMN .

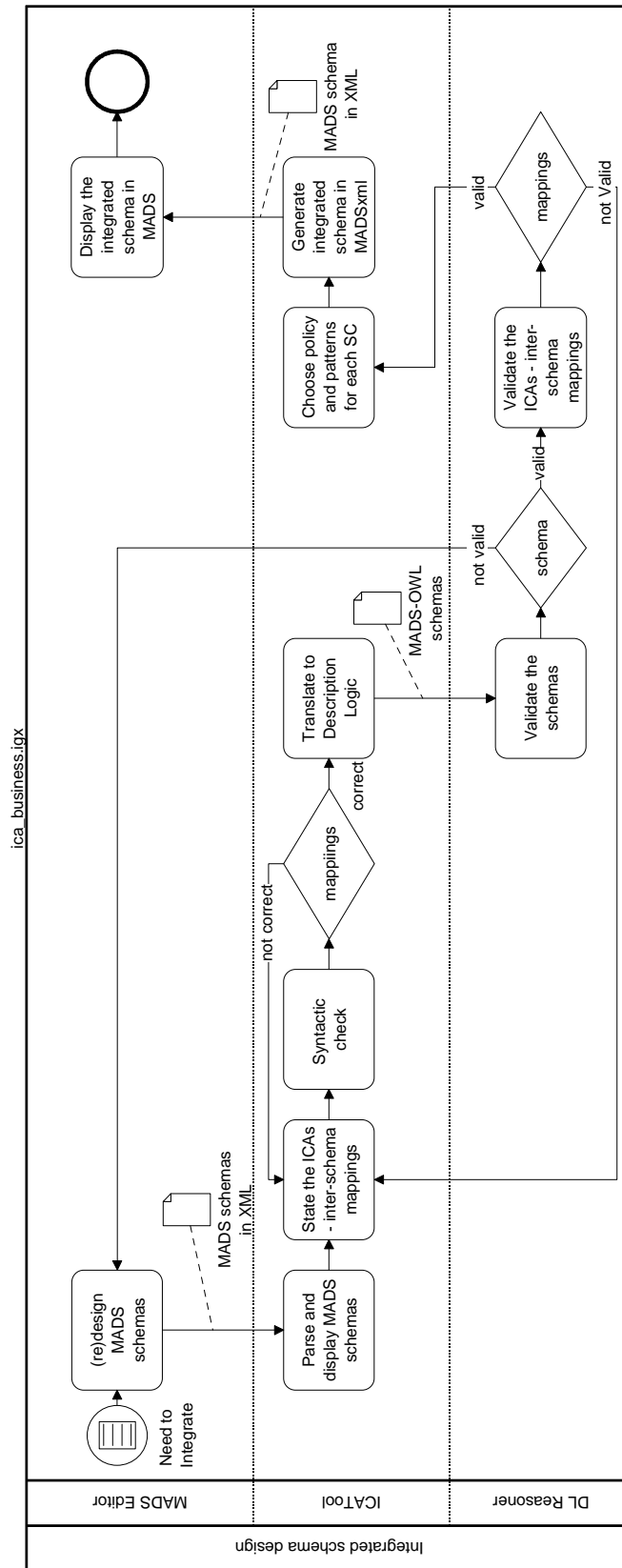


Figure 6.9: BP diagram for the ICATool.

6.3 Chapter Summary

In this chapter we have sketched the system models for the integration framework that includes several components. We have considered two modeling approaches, the UML and the SEAM approaches. Within both approaches we have defined the actors of the integration process together with their main functionalities. In UML, this corresponds to the use case; in SEAM, to the business organizational level model. The latter contains less elements (only one main functionality is shown), and therefore is more comprehensible for a wide circle of users (managers). Next level diagrams that model the processes performed by actors in the framework are the activity or robustness diagrams in UML, and the operation organizational level model in SEAM. On this level of modeling the advantage of SEAM approach is in its hierarchical structure and the unified notation. A SEAM operation organizational level diagram models a process showing its super and sibling processes using a uniform notation to show relationships between process actors. We note, that a SEAM operation model could be enhanced by choosing the BPMN for the process description within it. In UML, the notation for the activity and robustness diagrams is different from that of the use case, which requires from the model designer to memorize more modeling elements with their different representations in different diagrams. Moreover, an UML activity diagram does not show the data used in the process, while a SEAM operation model contains the object model of the process.

Chapter 7

Conclusions and Future Issues

7.1 Contributions of the Thesis

In this thesis we have addressed a complex problem of information integration. We conducted our research in the context of the projects of our laboratory, and we developed a specific integration method for spatio-temporal data. We have elaborated on a novel integration approach with validation that results in a spatio-temporal integrated schema that is flexibly constructed to meet the application requirements. The method is based on the expressive spatio-temporal conceptual model MADS, developed in our laboratory.

We have defined the MADS-based language for inter-schema mappings that allows explicit use of topological and synchronization operators to relate spatial and temporal object types. The language conforms to the data model and inherits its semantics. This correspondence language extends the MADS data model with integration capabilities, thus, enabling its prospective usage in large-scale collaborative projects. The data model and, therefore, the correspondence language are not limited to the spatio-temporal domain and efficiently treat thematic data.

We positioned our methodology as adhering to the federated information systems approach. Accordingly, our integration process results in an integrated schema that is called a *global* schema in a federated system. The integrated schema designer is guided towards constructing a global schema that corresponds to the application needs. Our distinctive vision on the integrated schema construction includes providing the designer with several structural policies and patterns that result in different integrated schema elements, an example is shown in Figure 3.27. The designer is not constrained to choose the same policy or pattern for all related populations, instead he/she can choose the most adapted structural solution for each population correspondence. The set of structural patterns defined for the integration process guarantees that there is always at least one patterns that can be applied to produce a valid integrated schema element.

Another original feature of our approach is that we have included and formalized the validation phase, where the set of the inter-schema mappings is checked for the

consistency, and the putative structural patterns are validated against the integrity constraints imposed on the source schemas. To validate user input, the source schemas together with the integrity constraints and the inter-schema mappings are translated into a description logic language. The translated model is then checked for satisfiability. A successful satisfiability check allows the schema designer to follow any of the integration policies, i.e., minimal, non-exhaustive, maximal, or preservation. In the case of a non-satisfiable model, only the preservation policy with the multiple representation structural patterns is a viable solution for the integrated schema.

We have considered the choice of a formalism for the validation phase of our method from two perspectives: theoretical and practical. From the theoretical aspect, we pursued the expressiveness of the formalism. We have shown that MADS conceptual schemas and inter-schema mappings can be expressed using an expressive description logic (DL) with a combined, spatial-temporal concrete domain. But we have faced the tradeoff between the expressivity and decidability, i.e., very expressive description logics are undecidable and therefore, there are no reasoning services developed for them. To practically implement the validation phase, we have chosen a DL-base language that is decidable and supported by an available reasoner. We implemented the MADS model semantics needed for reasoning about the inter-schema mappings in OWL-DL (Description Logic sub-language of the Ontology Web Language). Thus we have coupled two modeling approaches, i.e., conceptual modeling and modeling with ontologies in a hybrid integration process.

The OWL-DL provides neither for spatial nor for temporal support, thus, we have constructed a reference spatio-temporal ontology, MADS-OWL, that serves as the domain ontology and ensures the spatial and temporal constraints in user schemas. We have defined the translation from the MADS formalism to the OWL, stating the corresponding modeling concepts in each language. User ontologies expressed in MADS-OWL, can benefit from the reasoning services available for the OWL-DL ontologies. We have designed the MADS-OWL domain ontology in a way that the spatio-temporal features we need to validate could be captured by a non spatio-temporal reasoner.

To combine all the composite elements of our integration method together and explicitly model their functionality and the cooperations between them, we have chosen two system modeling approaches and designed high level system diagrams. We have shown that a hierarchical modeling approach with a unified notation results in a more expressive system model that can subsequently serve as the conceptual foundation for the integration framework.

7.2 Future Directions

Integration of several independently developed databases is a complex task, which is becoming increasingly important in the modern age information technology. In this

thesis we have considered several aspects of a specific problem of spatio-temporal data integration. We could not fulfil all the deficiencies in our domain of interest, and here we present some prospective developments for the issues we have considered in this thesis.

For the integration methodology. We have developed the MADS-OWL ontology to model MADS spatial and temporal dimensions in description logic. As we employed MADS representational dimension only for the final schema composition (i.e., in the preservation policy), this dimension is not fully implemented in MADS-OWL reference ontology. In particular, the MADS concept of multiple perceptions adds a great flexibility and expressivity to conceptual models, especially in the domain of geo-applications. Thus, we might expect that some of the source spatio-temporal schemas could already employ multiple perceptions. To extend our methodology to treating source schemas with multiple perceptions, a more precise translation of the MADS representational dimension into DL should be implemented.

For data integration support. In our thesis we considered the integration and validation approaches only for the schema level. Extension of the integration and validation services to the instance level would be an important extension of our approach. This would expand the scope of the application of our method beyond the federated systems, i.e., it would be applicable to the data warehouses management. For the reasoning support on the instance level, specific spatial and temporal reasoners should be added to the framework. What we have done manually for MADS topological relationships using the Protégé Axiom Language (cf. Figure 5.21), could be delegated to the specific spatial and temporal reasoners. Thanks to the MADS (and MADS-OWL) orthogonality in spatial, temporal and structural dimensions, we can decompose the model and the population in 3 subsets: spatial, temporal and thematic. Then we will apply 3 different machineries to reason about the network of temporal intervals (for instances with temporal extension), spatial network (for instances with spatial extension) and use description logic reasoner for ABox reasoning. If an instance is defined as spatial and temporal, then for each type of reasoning only relevant properties would be considered. To the best of our knowledge, an integration framework combining conceptual modeling and modeling with ontologies and supported by spatial and temporal reasoning services on the schema and the instance levels, has not been even conceived, let alone implemented.

For the integration framework. We have proposed a framework where the schema designer uses several tools to construct an integrated schema. The integrated schema design process is not automated due to the complex semantics of the application domain we consider. One of the improvements in the assisted schema design would be definition of user profiles to facilitate the choice of the integration policies and structural patterns. When the *schema population correspondences* are

stated, the ICATool would ask the designer to indicate a preferred integration policy for each of the related populations. If, after the validation phase, a policy is found applicable, the integrated schema element would be generated without the designer intervention. Otherwise, the designer will be prompted to change the policy. Note, that there is a preservation policy that can be applied even in the case of incompatible representations.

As we have shown in Section 4.2.4, the *matching rules* stated during the integration process can serve as the integrity constraints for an integrated database. Thus, providing the extension of the MADS data model with the integration capabilities, we should consider the translation of the inter-schema mappings from the conceptual to the logical level, e.g., to Oracle triggers. These triggers would inform the schema designer (or the database administrator) about the matching instances in the database. The translation rules should be coherent with already implemented translation procedures in the MADS translation module.

Bibliography

- [AF98] A. Artale and E. Franconi. A temporal description logic for reasoning about actions and plans. *Journal of Artificial Intelligence Research*, 1998.
- [AF01] A. Artale and E. Franconi. A survey of temporal extensions of description logics. *Annals of Mathematics and Artificial Intelligence (AMAI)*, 2001.
- [AKPT91] J.F. Allen, H.A. Kautz, R.N. Pelavin, and J.D. Tenenber, editors. *Temporal reasoning and planning*, chapter 1, pages 2–68. Morgan Kaufmann, 1991.
- [AL04] A. Artale and C. Lutz. A correspondence between temporal description logics. *Journal of Applied Non-Classical Logic*, 14(1-2):209–233, 2004.
- [All83] J. F. Allen. Maintaining knowledge about temporal intervals. *Communications of the ACM*, 26(11):832–843, 1983.
- [AM99] G. Aslan and D. McLeod. Semantic heterogeneity resolution in federated databases by metadata implantation and stepwise evolution. *VLDB Journal*, (8):120–132, 1999.
- [Amb00] S. Ambler. How to draw UML activity diagrams. <http://www-106.ibm.com>, 2000.
- [AOTT98] D. Abel, B. C. Ooi, K.-L. Tan, and S. H. Tan. Towards integrated geographical information processing. *International Journal of Geographical Information Science*, 12(4):353–371, June 1998.
- [BBB+98] O. T. Balovnev, A. Bergman, M. Breunig, A. B. Gremers, and S. Shumilov. A CORBA-based approach to data and system integration for 3D geoscientific application. In *Proceedings of the 8th International Conference on Spatial data Handling (SDH'98)*, pages 396–407, Vancouver, Canada, July 1998.

- [BBMHR91] A. Borgida, R. Brachman, D. McGuinness, and L. Halpern-Resnick. CLASSIC: A structural data model for objects. In *in Proc. of the 1989 ACM SIGMOD International Conference on Data*, pages 59–67, 1991.
- [BCdG01] D. Berardi, D. Calvanese, and G. de Giacomo. Reasoning on UML class diagrams using description logic based systems. In *Proc. of the KI'2001 Workshop on Applications of Description Logics*, 2001. CEUR Electronic Workshop Proceedings, <http://ceur-ws.org/Vol-44/>.
- [BCM⁺03] F. Baader, D. Calvanese, D. McGuinness, D. Nardi, and P. Patel-Schneider, editors. *The Description Logic Handbook: Theory, Implementation, and Applications*. Cambridge University Press, 2003.
- [BH91] F. Baader and P. Hanschke. A scheme for integrating concrete domains into concept languages. In *Proc. of the Twelfth International Joint Conference on Artificial Intelligence (IJCAI)*, pages 452–457, Sydney, Australia, 1991.
- [BHS03] F. Baader, I. Horrocks, and U. Sattler, editors. *Description Logics as Ontology Languages for the Semantic Web*. Lecture Notes in Artificial Intelligence. Springer-Verlag, 2003.
- [BLN86] C. Batini, M. Lenzerini, and S. Navathe. A comparative analysis of methodologies for database schema integration. *ACM Computing Survey*, 18(4):323–364, 1986.
- [BLR03] A. Borgia, M. Lenzerini, and R. Rosati. *The Description Logic Handbook*, chapter Description Logics for Databases, pages 462–484. Cambridge University Press, 2003.
- [BRW03] P. Balabko, I. Rychkova, and A. Wegmann. Operational ASM Semantics behind Graphical SEAM Notation. In *In Proc. of the DAIS/FMOODS Ph.D. workshop*, Paris, 2003.
- [BS85] R. J. Brachman and J. G. Schmolze. An overview of the KL-ONE knowledge representation system. *Cognitive Science*, 9(2):171–216, 1985.
- [CdGL97] D. Calvanese, G. de Giacomo, and M. Lenzerini. Conjunctive query containment in description logics with n-ary relations. In *Proc. of the 1997 Description Logic Workshop (DL'97)*, pages 5–9, 1997.
- [CdGL⁺98] D. Calvanese, G. de Giacomo, M. Lenzerini, D. Nardi, and R. Rosati. Information integration: Conceptual modeling and reasoning support. In *CoopIS 1998*, pages 280 – 291, 1998.

- [CdGL01] D. Calvanese, G. de Giacomo, and M. Lenzerini. A Framework for Ontology Integration. In *Proceedings of SWWS'01, The first Semantic Web Working Symposium Stanford University*, pages 303–306, California, USA, July 30 - August 1 2001.
- [CG05] D. Calvanese and G. De Giacomo. Data integration: A logic-based perspective. *AI Magazine*, 26(1):59–70, 2005.
- [CH01] A. G. Cohn and S. M. Hazarika. Qualitative spatial representation and reasoning: an overview. *Fundamenta Informaticae*, 46(1-2):1–29, 2001.
- [CL93] T. Catarci and M. Lenzerini. Representing and using interschema knowledge in cooperative information systems. *Journal of Intelligent and Cooperative Information Systems*, 2(4):375 – 398, 1993.
- [CLN98] D. Calvanese, M. Lenzerini, and D. Nardi. Description logics for conceptual data modeling. In Jan Chomicki and Gnter Saake, editors, *Logics for Databases and Information Systems*, pages 229–263. Kluwer Academic Publisher, 1998.
- [CLN99] D. Calvanese, M. Lenzerini, and D. Nardi. Unifying Class-Based Representation Formalisms. *Journal of Artificial Intelligence Research*, (11):199–240, 1999.
- [Coc00] A. Cockburn. *Writing Effective Use Cases*. Addison Wesley, 2000.
- [CPSV03] N. Cullot, C. Parent, S. Spaccapietra, and C. Vangenot. Ontologies: A Contribution to the DL/DB Debate. In *In Proc. of the 1st International Workshop on the Semantic Web and Databases, 29th International Conference on Very Large Data Bases*, Berlin Germany, September 7-8 2003.
- [CR99] J. Chomicki and P. Z. Revesz. Constraint-based interoperability of spatio temporal databases. *GeoInformatica*, 3(3):211–244, September 1999.
- [dic] Online Dictionary of Computer Science.
<http://burks.bton.ac.uk/burks/foldoc/>.
- [DLD⁺04] R. Dhamankar, Y. Lee, A. Doan, A. Halevy, and P. Domingos. iMAP: Discovering Complex Semantic Matches between Database Schemas. In *Proc. of the 2004 ACM SIGMOD international conference on Management of data*, pages 383 – 394, Paris, France, June 13-18 2004. ACM Press.

- [DMDH02] A. Doan, J. Madhavan, P. Domingos, and A. Halevy. Learning to map between ontologies on the Semantic Web. In *Proc. of International WWW conference, WWW'02*, Honolulu, Hawaii, USA, May 7-11 2002.
- [DMDH04] A. Doan, J. Madhavan, P. Domingos, and A. Halevy. Ontology matching: A machine learning approach. In S. Staab and R. Studer, editors, *Handbook on Ontologies in Information Systems*, pages 397–416. Springer-Verlag, 2004.
- [Don02] P. Donini. *NOMADS: a Spatio-temporal Data Model Supporting Multi-instantiation*. PhD thesis, École Polytechnique Fédérale de Lausanne, 2002.
- [DPS98] T. Devogele, C. Parent, and S. Spaccapietra. On spatial database integration. *International Journal of Geographic Information Systems, Special Issue on System Integration*, 3(12):335–352, 1998.
- [Dup94] Y. Dupont. Resolving Fragmentation Conflicts in Schema Integration. In P. Loucopoulos, editor, *13th International Conference on the Entity-Relationship Approach, ER'94*, volume 881 of *LNCS*, pages 513–532, Manchester, U.K., December 13-16 1994. Springer.
- [EP90] A. Elmagarmid and C. Pu. Guest editors' introduction to the special issue on heterogeneous databases. *ACM Computing Survey*, 22(3):175–178, 1990.
- [ES04] M. Ehrig and Y. Sure. Ontology mapping - an integrated approach. In *Proc. of the ESWS 2004*, pages 76–91, Heraklion, Crete, Greece, May 2004.
- [FaC] <http://owl.man.ac.uk/factplusplus/>.
- [FDC03] F. Fonseca, C. Davis, and C. Câmara. Bridging ontologies and conceptual schema in geographical information integration. *Geoinformatica*, 7(4):355–378, 2003.
- [FEAC02] F. Fonseca, M. Egenhofer, P. Agouri, and C. Câmara. Using ontologies for integrated geographic information systems. *Transactions in GIS*, 6(3):231–257, 2002.
- [FN00] E. Franconi and G. Ng. The i.com tool for intelligent conceptual modelling. In *In Proc. of the 7th Intl. Workshop on Knowledge Representation meets Databases (KRDB'00)*, Berlin, Germany, August 2000.
- [FPNB99] J. Fowler, B. Perry, M. Nodine, and B. Bargmeyer. Agent-Based Semantic Interoperability in InfoSleuth. *ACM SIGMOD Records*, 28(1):60–67, March 1999.

- [GEFK99] M. Goodchild, M. Egenhofer, R. Fegeas, and C. Kottman, editors. *Interoperating Geographic Information Systems*. Kluwer Academic Publishers, 1999.
- [GMZB99] A. Gupta, R. Marciano, I. Zaslavsky, and C. Baru. Integrating GIS and Imagery through XML-based Information Mediator. In P. Agouris and A. Stefanidis, editors, *International Workshop on Integrated Spatial Databases: Digital Images and GIS (ISD'99)*, volume 1737 of *Lecture Notes in Computer Science*. Springer, Portland, Maine, USA, June 1999.
- [GN02] A. Gerevini and B. Nebel. Qualitative spatio-temporal reasoning with RCC-8 and Allen's interval calculus: Computational complexity. In *Proceedings of ECAI'2002*, pages 312–316. IOS Press, 2002.
- [Gua98] N. Guarino. *Information Extraction: A Multidisciplinary Approach to an Emerging Information Technology*, chapter Semantic Matching: Formal Ontological Distinctions for Information Organisation, Extraction, and Integration, pages 139–170. Springer-Verlag, 1998. M.T. Paziienza.
- [GW00a] N. Guarino and Ch. Welty. Ontological analysis of taxonomic relationships. In A. Lander and V. Storey, editors, *Proceedings of ER-2000: The 19th International Conference on Conceptual Modeling.*, LNCS. Springer-Verlag, October 2000.
- [GW00b] N. Guarino and Ch. Welty. Towards a methodology for ontology based model engineering. In J. Bezivin and J. Ernst, editors, *Proceedings of the ECOOP-2000 Workshop on Model Engineering*, June 2000.
- [HG02] F. Hakimpour and A. Geppert. Global schema generation using formal ontologies. In *Proc. of the ER2002*, LNCS 2503, pages 307–321, 2002.
- [HIMT03] A. Y. Halevy, Z. G. Ives, P. Mork, and I. Tatarinov. Piazza: Data Management Infrastructure for Semantic Web Applications. In *Proc. of International WWW conference, WWW'03*, pages 556–567, 2003.
- [HLM99] V. Haarslev, C. Lutz, and R. Möller. A description logic with concrete domains and a role-forming predicate operator. *Journal of Logic and Computation*, 9(3), 1999.
- [Hor98] I. Horrocks. Using an expressive description logic: FaCT or Fiction? In A. G. Cohn, L. Schubert, and S. C. Shapiro, editors, *Principles of Knowledge Representation and Reasoning: Proceedings of the Sixth International Conference (KR'98)*, pages 636–647, San Francisco, California, June 1998. Morgan Kaufmann Publishers.

- [HPPSH05] I. Horrocks, B. Parsia, P. Patel-Schneider, and J. Hendler. Semantic web architecture: Stack or two towers?, 2005.
- [HST00] I. Horrocks, U. Sattler, and S. Tobies. Practical reasoning for expressive description logics. *Logic Journal of the IGPL*, 8(3):161 – 180, 2000.
- [JM00] N. Juristo and A. M. Moreno. Introductory paper: Reflection on conceptual modeling. *Data & Knowledge Engineering*, (33):103–117, July 2000.
- [KLWZ04] O. Kutz, C. Lutz, F. Wolter, and M. Zakharyashev. ϵ -connections of abstract description systems. *Artif. Intell.*, 156(1):1–73, 2004.
- [Lut02] C. Lutz. *The Complexity of Reasoning with Concrete Domains*. PhD thesis, Teaching and Research Area for Theoretical Computer Science, RWTH Aachen, 2002.
- [Lut03] C. Lutz. Description logics with concrete domains—a survey. In *Advances in Modal Logics Volume 4*. King’s College Publications, 2003.
- [LW04] L.-S. Lê and A. Wegmann. Meta-model for Object-Oriented Hierarchical Systems. Technical report, Laboratory of Systemic Modeling Swiss Federal Institute of Technology, Lausanne EPFL-IC-LAMS, 2004.
- [Mac94] R. M. MacGregor. A description classifier for the predicate calculus. In *in Proceedings of the Twelfth National Conference on Artificial Intelligence, (AAAI 94)*, pages 213–220, 1994.
- [MBDH02] J. Madhavan, P. Bernstein, P. Domingos, and A. Halevy. Representing and reasoning about mappings between domain models. In *Proc. of the AAAI Eighteenth National Conference on Artificial Intelligence*, pages 80 – 86, Edmonton, Alberta, Canada, 2002. American Association for Artificial Intelligence.
- [Mur02] MurMur consortium. MurMur project: Multi-representations and Multiple resolution in geographic databases. <http://lbdwww.epfl.ch/e/MurMur>, 2002.
- [OCEF05] M. M. Ortiz, D. Calvanese, Th. Eiter, and E. Franconi. Data Complexity of Answering Conjunctive Queries over *SHIQ* Knowledge Bases. Technical report, Free University of Bozen-Bolzano, November 2005.
- [OGI] OpenGIS Consortium. <http://www.opengis.org>.
- [OWL] <http://www.w3.org/TR/owl-semantics/>.

- [PAL] Protégé axiom language. <http://protege.stanford.edu/plugins/paltabs/pal-documentation/index.htm>.
- [Pel91] C. Peltason. The BACK system - an overview. *SIGART Bulletin*, 2(3):114–119, 1991.
- [PRO] <http://protege.stanford.edu/>.
- [PS98] C. Parent and S. Spaccapietra. Database integration: the key to the data interoperability. Technical report, EPFL, CH-1015 Lausanne, 1998.
- [PSV05] C. Parent, S. Spaccapietra, and C. Vangenot. Multiple representations in geographic databases. In *Proceedings of the International Conference, Somewhere, Somewhen 2005*.
- [PSZ99] C. Parent, S. Spaccapietra, and E. Zimanyi. Spatio-Temporal Conceptual Models: Data Structures + Space + Time. In *7th ACM Symposium on Advances in GIS, Kansas City, Kansas, Kansas City, Kansas, November 5-6 1999*.
- [PSZ05] C. Parent, S. Spaccapietra, and E. Zimányi. The MurMur Project: Modeling and Querying Multi-Representation Spatio-Temporal Databases. *Information Systems*, 2005. In Press.
- [PSZ06] C. Parent, S. Spaccapietra, and E. Zimányi. *Conceptual Modeling for Traditional and Spatio-Temporal Applications: The MADS Approach*. Springer-Verlag, 2006. Not yet published. Available: May 15, 2006.
- [QL94] X. Qian and T. F. Lunt. Semantic interoperation: A query mediation approach. Technical Report SRI-CSL-94-02, Computer Science Laboratory, SRI International, April 1994.
- [RAC] <http://www.racer-systems.com/>.
- [RCC92] D. A. Randell, Z. Cui, and A. G. Cohn. A spatial logic based on regions and connection. In *Proceedings of the 3rd International Conference on Knowledge Representation and Reasoning (KR'92)*. Morgan Kaufmann, 92.
- [RS01] K. Rosenberg and K. Scott. *Applying Use Case Driven Object Modeling with UML*. Addison Wesley, 2001. An Annotated e-Commerce Example.
- [Sat96] U. Sattler. A concept language extended with different kinds of transitive roles. In G. Görz and S. Hölldobler, editors, *20. Deutsche Jahrestagung für Künstliche Intelligenz*, number 1137 in Lecture Notes in Artificial Intelligence. Springer Verlag, 1996.

- [Sem05] SemanticWeb. KAON2. <http://kaon2.semanticweb.org/>, 2005.
- [She99] A. Sheth. *Changing Focus on Interoperability in Information Systems: from System, Syntax, Structure to Semantics*, chapter in [GEFK99]. Kluwer Academic Publishers, 1999.
- [SL90] A. Sheth and J. Larson. Federated database systems for managing distributed, heterogeneous, and autonomous databases. *ACM Computer Surveys*, (2):183–236, September 1990.
- [SMS02] A. Sotnykova, S. Monties, and S. Spaccapietra. Semantic integration in MADS conceptual model. In H. Bestougeff and B. Thuraisingham, editors, *Heterogeneous Information Exchange and Organizational Hubs*. Kluwer, 2002.
- [SPV00] S. Spaccapietra, C. Parent, and C. Vangenot. GIS Databases: From Multiscale to MultiRepresentation. In B.Y.Choueiry and T.Walsh, editors, *Proceedings 4th International Symposium, SARA-2000*, volume 1864 of *LNAI*, Horseshoe Bay, Texas, USA, July 26-29 2000. Springer-Verlag.
- [SPVC04] S. Spaccapietra, C. Parent, C. Vangenot, and N. Cullot. On using conceptual modeling for ontologies. In *In Proceedings of the International Workshop on the Intelligent Networked and Mobile Systems, Ontologies for Networked Systems (ONS) track, located at the 5th International Conference on Web Information Systems Engineering WISE 2004*, Brisbane, Australia, 22nd-24th November 2004.
- [TH04] D. Tsarkov and I. Horrocks. Efficient Reasoning with Range and Domain Constraints. In *Proc. of the International Workshop on Description Logics (DL-2004)*, Whistler, British Columbia, Canada, June 6-8 2004.
- [Tha00] B. Thalheim. *Entity-relationship modeling: foundations of database technology*. Springer-Verlag, 2000.
- [Van01] C. Vangenot. *Multi-représentation dans les bases de données géographiques*. PhD thesis, École Polytechnique Fédérale de Lausanne, 2001.
- [Weg03] A. Wegmann. On the Systemic Enterprise Architecture Methodology (SEAM). In *ICEIS (3)*, pages 483–490, 2003.
- [Wes02] M. Wessel. On spatial reasoning with description logics - position paper. In *Proc. of the International Workshop in Description Logics 2002 (DL2002)*, Toulouse, France, April 19-21 2002.

-
- [Whi04] S. A. White. *Introduction to BPMN*. IBM, 2004.
- [Wie92] G. Wiederhold. Mediators is the architecture of future information systems. *The IEEE Computer Magazine*, March 1992.
- [WVV⁺01] H. Wache, T. Vögele, U. Visser, H. Stuckenschmidt, G. Schuster, H. Neumann, and S. Hübner. Ontology-Based Integration of Information - A Survey of Existing Approaches. In *Proceedings of the IJCAI-01 Workshop on Ontologies and Information Sharing*, Seattle, USA, August 4-5 2001.
- [WZ00] F. Wolter and M. Zakharyashev. Spatio-temporal representation and reasoning based on RCC-8. In *Proceedings of the International Conference on Principles of Knowledge Representation and Reasoning, KR2000*, 2000.

Sotnykova Anastasiya

24 December 1973, single, Ukrainian

Database Laboratory, EPFL-IC-IIF-LBD, Station 14, CH-1015 Lausanne, Switzerland
Anastasiya.Sotnykova@epfl.ch, Tel: +41 78 656 87 18



Experience

2002 – 2005

Research Assistant, PhD candidate (EPFL, Switzerland)

PhD thesis on integration of geographical databases is defended in December 2005.

- led final projects for graduate students – conceptual modeling, UML, Ontology Web Language (OWL), Apache Cocoon Framework, Tomcat, Xindice XML DBMS
- prepared and taught a graduate class on database integration – geographical data modeling, MapInfo;
- published 4 conference and 1 journal paper

2000 – 2003

Principal participant in a project for the Ministry of environment and the Planning department (Luxembourg)

- remodeled legacy environmental data for a more efficient usage – specific database integration methodology for geographical data, UML, Java, JavaCC
- trained staff members in conceptual modeling of spatio-temporal information – conceptual data modeling approach for geographical data (MADS)

1997 – now

Teaching Assistant (EPFL, Switzerland; Bilkent University, Turkey)

- taught undergraduate courses on Operating Systems and OO Programming Languages
- prepared and taught graduate courses on Database Design and Advanced Databases – conceptual data modeling, Oracle 9i, 10g, Oracle XML, Oracle Spatial, SQL, XML, XPath, XQuery

1996

Computer Engineer (Kharkov State Polytechnic University, Ukraine)

- designed and developed a decision support system for liver surgery within a joint project with Kharkov Medical University – data mining, Clipper DBMS, C++
- designed and supported exercise and project sessions for department courses – Assembler, Pascal, FoxPro
- translated technical documentations for department usage

1994 – 1996

Freelance Developer (Ukraine)

- developed specific management systems for several companies such as a
 - client and contract management for an insurance company serving 1000 clients
 - stock handling for a bearing trade company
 - client management for a small tourist agency

Education

2005

PhD in Computer Science, EPFL, Lausanne, Switzerland

1999

Pre-doctoral school certificate, EPFL, Lausanne, Switzerland

1998

M.S. in Computer Engineering and Information Science, Bilkent University, Turkey

1994, 1996

B.S. and M.S. in Computer Engineering, system analyst specialization. Kharkov State Polytechnic University, Ukraine. Graduated with High Honour

Languages

Russian, Ukrainian, English, French, Italian (basic)

Personal Interests

Summer and winter mountain sports, biking, tango dancing

Publications

A. Sotnykova, N. Cullot, and C. Vangenot.

Spatio-temporal Schema Integration with Validation: A Practical Approach, in R. Meersman et al. (Eds.): OTM Workshops 2005, LNCS 3762, 2005

A. Sotnykova, C. Vangenot, N. Cullot, N. Bennacer, and M-A. Afaure.

Semantic Mappings in Description Logics for Spatio-Temporal Database Schema Integration
Journal on Data Semantics III, LNCS 3534, 2005

N. Bennacer, M.-A. Afaure, N. Cullot, A. Sotnykova. Representing and Reasoning for Spatiotemporal Ontology Integration. OTM Workshops, 2004

Anastasiya Sotnykova: Geodata Interoperation via Semantic Correspondences. CoopIS / DOA / ODBASE , 2002

A. Sotnykova, S. Monties, S. Spaccapietra, Semantic Integration in MADS Conceptual Model

In H. Bestougeff, B. Thuraisingham, editors, In Heterogeneous Information Exchange and Organizational Hubs, Kluwer, 2002