Roma Tre University
Ph.D. in Computer Science and Engineering

Modeling and interoperability: a high level perspective

Pierluigi Del Nostro

Modeling and interoperability: a high level perspective

A thesis presented by
Pierluigi Del Nostro
in partial fulfillment of the requirements for the degree of
Doctor of Philosophy
in Computer Science and Engineering

Roma Tre University
Dept. of Informatics and Automation

February 2009

Committee:
Prof. Paolo Atzeni

Reviewers:
Prof. Ernest Teniente
Prof. Martine Collard

Abstract

This thesis tackles modeling and interoperability issues in different contexts. We started by studying different Semantic Web models with the goal of translating from one to another by means of a model independent approach. The metamodel approach that we follow is called MIDST and is based on the concept of *supermodel*, a generic model that we use to describe other models. We have extended this approach, to allow the interoperability between Semantic Web formalisms. MIDST leverage on a relational dictionary that we have exploited as a repository for RDF documents. The logical organization that we have defined, together with tuning techniques at the physical level, allows us to obtain a framework for storing and querying RDF, that produced great results in terms of performance and scalability. Following the experience gained in modeling Semantic Web models, we have produced a new enhancement in MIDST expressivity, allowing the interchange of information between ontologies and databases. Changing context, this thesis finally describe a framework for the modeling of time in data-intensive Web sites. We here developed a tool that allows to automatically generate the Web site as a consequence of the design choices.

Acknowledgments

I first would thank Prof. Paolo Atzeni, for the things I had learnt working together. Thanks to my colleagues and friends Stefano Paolozzi, Giorgio Gianforme and Roberto De Virgilio with whom I have shared the PhD experience. I will never thank enough my family, for the support that gave me. A special thought to Michela, for choosing to share her life with me. Many thanks to my friends Luigi Arlotta and Alessandro Kayed, that helped me during the overburden periods of my work.

Contents

Contents **viii**

List of Tables **x**

List of Figures **xi**

1 Introduction **1**

2 Management of heterogeneous models **5**
2.1 The context 6
2.2 The MIDST approach 7
2.3 Model independence 7
2.4 The relational dictionary 11
2.5 Conclusions 14

3 Translation between Semantic Web annotation formalisms **17**
3.1 RDF 18
3.2 Topic Maps 28
3.3 Models comparison 35
3.4 Related works 36
3.5 Extending MIDST to Semantic Annotations 54
3.6 Translation between RDF and Topic Maps 60
3.7 Conclusions 65

4 A Scalable and Extensible Framework for the Management of RDF data **67**
4.1 Introduction 68
4.2 Running Example 70

4.3 Related Works 72
4.4 Management of RDF data 75
4.5 Experimental Results 86
4.6 Conclusions 92

5 Living with ontologies and databases 95
5.1 Introduction 95
5.2 Related Work 97
5.3 Extending MIDST supermodel to Semantic Web 98
5.4 From OWL ontology to Relational Database 101
5.5 From Relational Database to OWL Ontology 108
5.6 Conclusions 112

6 Temporal aspects for data intensive Web sites 113
6.1 Introduction 113
6.2 The Araneus models and methodology 116
6.3 Models for the management of temporal aspects of Web sites . 118
6.4 CMS support to T-Araneus 130
6.5 An Example Application 135
6.6 Conclusions 139

Conclusion 141

Appendices 143

Appendix A 145

Appendix B 147

List of Tables

3.1 Moore RDF-Topic Maps mapping 41
3.2 Relational table Person 55

List of Figures

2.1 MIDST translation process. 8
2.2 Basic Datalog rules. 10
2.3 Relationships between models, constructs, meta-constructs 11
2.4 A portion of the MIDST relational dictionary 12
2.5 ER model dictionary 13
2.6 Two samples of ER schemas 14
2.7 Partition of the MIDST abstraction layers 14

3.1 RDF Example 19
3.2 RDF Graph Example 21
3.3 Statements in an RDF Graph representation 21
3.4 Structured information representation in RDF 23
3.5 Example of an RDF collection 27
3.6 RDF example for the relation Person 56
3.7 RDF example representing only name and surname 57
3.8 The supermodel's constructs used to represent RDF 60
3.9 The supermodel's constructs used to represent Topic Maps 60
3.10 RDF Example 62
3.11 MIDST Relational dictionary storing an RDF document 63
3.12 Topic Maps example 65
3.13 Resulting RDF graph 65

4.1 RDF classes and triple instances 71
4.2 The *vertical partitioning* tables for the running example. 74
4.3 RDF data Management with a 3-layer model 76
4.4 The three levels of abstraction 76
4.5 The M_{RDF} model 79

4.6 Conceptual representation of the running example 80
4.7 Logical representation of our model 81
4.8 Physical Organization of the running example 82
4.9 Logical representation of Containers 85
4.10 Performance comparison between triple-vertical-swim approaches . 90
4.11 Query Performance as number of triples scale 92
4.12 Maintenance Performance as number of triples scale 93

5.1 A portion of MIDST supermodel. 100
5.2 Correspondences between the OWL model and the supermodel. . . 103
5.3 MIDST Tables. 105
5.4 Correspondences between the relational model and the supermodel. 109

6.1 The Araneus design process 117
6.2 The example of ER schema 117
6.3 The example of N-ER schema 118
6.4 The example of ADM schema 119
6.5 Temporal notation example 121
6.6 Versions for the TeacherPage 123
6.7 Versions for the CoursePage 124
6.8 Two examples of temporal navigation 125
6.9 The TARGET CHANGED feature 128
6.10 The TARGET CHANGED feature along a path 128
6.11 The SIMPLE VERSION STRUCTURE pattern 129
6.12 The LIST VERSION STRUCTURE pattern 129
6.13 The CHAIN VERSION STRUCTURE pattern 129
6.14 The SUMMARY VERSION STRUCTURE pattern 129
6.15 The tree of available choices 130
6.16 Workflow model 133
6.17 Architecture of the system. 134
6.18 A screenshot of the CASE tool design interface. 136
6.19 The example T-ER schema 137
6.20 Temporal features in the N-ER model 137
6.21 A T-ADM page scheme. 138

A.1 The supermodel class diagram. 146

Chapter 1

Introduction

Modeling is a major activity in the database field. In our research group we are concentrating on two branches of interest concerning interoperability between heterogeneous models and the modeling of Web site aspects. In these two contexts my research activity has specifically produced contributions on Semantic Web models and Temporal Web sites, respectively.

Semantic Web is based on the idea of assigning a shared, unambiguous, machine computable meaning to the information, by means of descriptions of data. To aim this goal many models have been defined, from the simpler language to assign meta-data to the more sophisticated ontologies. In this work we give our contribution in the reconciliation of this heterogeneity, providing a means that ease the interoperability between different semantic representations. We pursue this goal by the use of a metamodelling technique and a methodology, that allows to define a high level translation process that is independent from the specific model. The approach is part of a wider project MIDST (Model Independent Schema and Data Translation), developed by the Database group of the Roma Tre University of Rome, that tackles many aspects of the model management area. We define a generic model (that we call *supermodel*) by means of which is possible to describe the other models capturing the structural aspects. By means of rules, written in a Datalog variant, that act on our meta-representations, the user can flexibly choose how to perform the translation between a source and a target model in a generic manner.

We started our experiments with the translation between the two datamodels, RDF and Topic Maps, for which many approaches are presented in the literature. After the study of the nature and the peculiarities of the two models,

we have underlined which are the sharing points and the differences. We have then applied our approach, translating from RDF to Topic Maps and viceversa, with specific attention in reducing the likely loss of semantic that can occur when translating between different models with different power of expressivity. Interoperability between RDF and Topic Maps allows different groups, using the two formalisms, to cooperate sharing the knowledge. Moreover, if one of the two formalisms, let's say RDF, become a universal standard, all the knowledge created by means of Topic Maps will not be lost but reused (almost partially).

Beyond the translation issues that regard the heterogeneity of Semantic Web models, we have faced with another problem of the Semantic Web community, related to the growth of meta-data over the Web. Data-sets of millions of RDF triple, describing different contexts, are publicly available on the internet. Many proposals exist on how to store, manage and query large RDF data-sets, exploiting a relational database, starting from simple triple representation, where data are organized in a unique table of triples, to some evolutions that use different logical organization of data. The triple storage model is obviously unsuitable to scale with the number of triples, due to the multitude of self joins needed; its extreme simplicity goes to the detriment of performance. The solution adopted by most of the recent proposals in this sense, consists of providing logical and physical partitioning of the triple data to increase the performance of storing and querying. After studying the state of the art of these proposals, we have implemented the ones that we considered the most significant and tested them against some representative queries. Experimenting the translations between Semantic Web formalisms, we used a relational database to store RDF data with the goal of translating them to Topic maps, then we asked ourselves if the storage model that we exploit, could be also suitable to reach good performances in querying and if the approach could be scalable. We have tested our approach with the same data-set and queries used to test the other approaches and the results comparison places our approach in the top both for the query response time and the scalability.

The positive results that we gathered in the aforementioned scenarios suggest us to expand the application of our approach. Starting from the translation between formalisms belonging to the same world, the Semantic Web, we have come to the challenge of translating between different worlds, namely Semantic Web and databases. More specifically we have studied the translation between OWL ontologies and relational databases. This context has been also studied intensively in the research community due to the variety of applications it involves. The two worlds have been conceived with different purposes: databases are used to efficiently and effectively store and query data, whereas

the Semantic Web aims at giving a unique interpretation of the meaning of data. Semantic Web and relational databases live together in an evolving Web where always more data intensive sites exist and where meta-data and ontologies are diffusing to increment the precision of the searches and to allow the share and reuse of knowledge. In my opinion each of the technologies should be used for the purpose they are conceived for, interacting, to let one exploit the potentiality of the other. Thus we can employ databases to store large ontologies or meta-data documents, while ontologies can be used to automatically enrich relational data to be published on the web with a semantic.

The other area of interest of my research activity regards the management of temporal aspects in data-intensive Web sites. A Web site is called data-intensive when its main purpose is the publishing of a large amount of data and we consider data stored in relational databases. In our group a tool to automatically generate a Web site, starting from a relational database repository, has been developed. It is based on a design process where the site structure and its content are described by means of different models at different levels of abstraction. We have enhanced these models with new constructs to manage temporal aspects both at data and pages level, leveraging on the studies made on temporal databases. With this tool we enable the site designer to express design choices about how to manage the time coordinate and the content versions, along the design process. At the end of the process, the Web site and a CMS system with features to manage time, are automatically generated by the tool, together with the database schema that captures the time coordinate.

In this document we will illustrate which are the contributions that we produced in these areas, tackling different topics with the same philosophy of finding solutions basing on a modeling approach.

The rest of the document is organized as follows: in chapter 2 an introduction to the MIDST approach is given; chapter 3 illustrates the translation between RDF and Topic Maps; chapter 4 shows how our storage model reveals a means for the efficient and scalable management of RDF data; in chapter 5 the translation between OWL and relational database is described; chapter 6 presents the work about the management of temporal data intensive Web sites.

Chapter 2

Management of heterogeneous models

In this chapter we will illustrate MIDST, an implementation of the model management operator ModelGen, which translates schemas and data from one model to another [**?**].

To manage heterogeneous data, many applications need to translate data and their descriptions from one model (i.e. data model) to another. The common requirement of this interoperability scenario is the ability to translate schemas and data across heterogeneous models.

To reconciliate this variety of representation standards, what we need is an approach that is generic across models, and can handle the peculiarities of each model.

The approach we propose here translates schemas from a model to another, within a predefined, but large and extensible, set of models: given a source schema *S* expressed in a source model, and a target model *TM*, it generates a schema *S'* expressed in *TM* that is "equivalent" to *S*. The different models are precisely described by using a metamodel representation. Translations are expressesed as Datalog rules and exposes the source and target of the translation in a generic relational dictionary. This makes the translation transparent, easy to customize and model-independent.

2.1 The context

The need of share and reuse of data among different applications implies the arise of issues related to the heterogeneity of data modeling and description standards that are so far diffused in the ICT community. Many organizations developed their own models and often the same model appear in a variety of declinations. Taking into account the database area, the evolution of technologies constantly enhance heterogeneity and therefore more need for translations. The diffusion of XML as an interchange standard lead database vendors to consider the introduction of functionality to directly manage XML data and to transform flat Relational data into nested XML and vice versa. The semantic Web context, even if younger than Databases, do not profit from the past experience. Many languages for the management and the representation of knowledge have been developed with different expressivity characteristics. RDF, Topic Maps, the various OWL declinations, TMCL, RDF(S), are only some, from the most popular, formalisms developed with the purpose of representing knowledge from the simpler meta-data to the ontologies. And the things got worst considering that the different serialization languages available for the same data-model. RDF, for example, can be serialized by using RDF/XML, Notation 3 or others, less known. Moreover, the two worlds, databases and semantic Web, compenetrate always more.

It should be clear that there is a growing need for translations in different areas. This problems belongs to the larger context Bernstein [**?**] termed model management, an approach to meta data management that considers schemas as the primary objects of interest and proposes a set of operators to manipulate them. In this thesis we consider the ModelGen operator [**?**], defined as follows: given two models M_1 and M_2 and a schema S_1 of M_1, ModelGen translates S_1 into a schema S_2 of M_2 that properly represents S_1.

Along with the schemas, if the instance level is of interest, the approach is applicable to data as well: given a database instance I_1 of S_1, the extended operator produces an instance I_2 of S_2 that has the same information content as I_1. As there are many different models, what we need is an approach that is generic across models and can handle the idiosyncrasies of each model. The main goal is to have one single approach that works over a variety of models, despite developing specific translations for each pair of models.

2.2 The MIDST approach

The main concept over which we build our approach is the *metamodel*. A metamodel is a set of generic constructs that can be used to define models, as a consequence a model is an instance of the metamodel as well as a schema is an instance of a model. We leverage on Hull and Kings intuition [**?**] that the constructs used in most known models can be expressed by a limited set of generic (i.e., model independent) metaconstructs: lexical, abstract, aggregation, generalization, function. We therefore define a metamodel by means of a set of generic metaconstructs. Each model is defined by its constructs and the metaconstructs they refer to. With respect to metaconstructs we can define some popular models as follows:

Entity Relationship model (i) abstracts (the entities), (ii) aggregations of abstracts (relationships), and (iii) lexicals (attributes of entities and, in most versions of the model, of relationships);

Object Oriented model (i) abstracts (classes), (ii) reference attributes for abstracts, which are essentially functions from abstracts to abstracts, and (iii) lexicals (fields or properties of classes);

Relational model (i) aggregations of lexicals (tables), (ii) components of aggregations (columns), which can participate in keys, (iii) foreign keys defined over aggregations and lexicals;

RDF (i) abstracts (resources), (ii) abstract attributes of abstracts (properties), (iii) foreign keys defined over aggregations and lexicals.

Models and schemas descriptions are maintained in a relational dictionary and references are used to store the connections between constructs. In section 2.4 some specific aspects of the dictionary implementation are illustrated.

2.3 Model independence

To implement an efficient translation system it is necessary to define a translation technique that can address the greatest possible number of models. For this reason MIDST includes the concept of *supermodel*, i.e. a model that contains all other models, more specifically, a model that contains the constructs through which all possible models are defined. Indeed, each model can be represented as a set of *meta-constructs* belonging to a more general model, which

is the *supermodel*. Thus, each model is a specialization of the supermodel and a schema in any model is also a schema in the supermodel, apart from the specific names used for constructs. In Appendix A, Fig.A.1 illustrate a Class Diagram representing the supermodel as it was before the modifications produced during my research activity.

The supermodel acts as a "pivot" model, so that it is sufficient to have translations from each model to and from the supermodel, rather than translations for every pair of models. The benefit, in terms of computational complexity, is that the number of translations needed is linear instead of quadratic. Moreover, since every schema in any model is also a schema of the supermodel, the only needed translations are those within the supermodel: a translation is composed of (a) a copy (with construct renaming) from the source model into the supermodel; (b) an actual transformation within the supermodel, whose output includes only constructs allowed in the target model; (c) another copy (again with renaming into the target model).

The whole process is sketched in Fig. 2.1

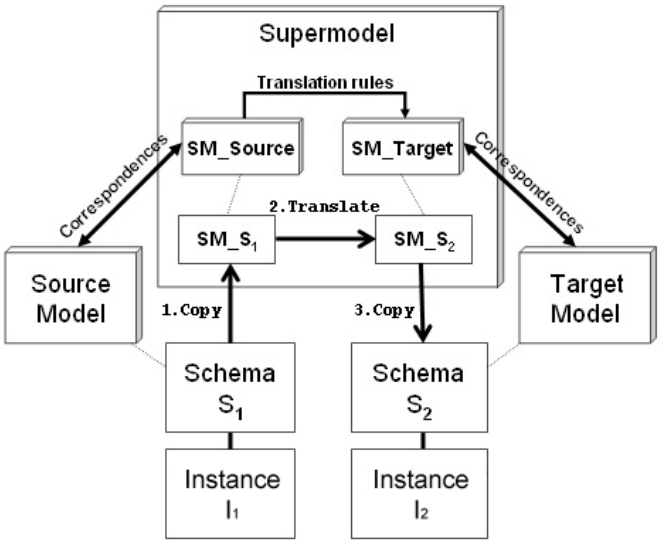


Figure 2.1: MIDST translation process.

Another relevant benefit of exploiting the supermodel is that the common

aspects of models are emphasized. If two source models share a construct, then their translations towards similar target models could share a portion of the translation as well. In our approach, we follow this observation by defining elementary (or basic) translations that refer to single constructs (or even specific variants thereof). Then, actual translations are specified as compositions of basic ones, with significant reuse. For example, assume we have as the source an ER model with binary relationships (with attributes) and no generalizations and as the target a simple OO model. To perform the task, we would first translate the source schema by renaming constructs into their corresponding homologous elements (abstracts, binary aggregations, lexicals, generalizations) in the supermodel and then apply the following steps:

1. eliminate attributes from aggregations of abstracts;
2. eliminate many-to-many aggregations of abstracts;
3. replace aggregations of abstracts with references;
4. eliminate generalizations (introducing new references);
5. eliminate generalizations (introducing new aggregations of abstracts);
6. replace aggregations of abstracts with references;
7. replace abstracts and their aggregations with aggregations of lexicals;
8. replace abstracts and references with aggregations of lexicals.

Translation rules are written in a Datalog variant with Skolem functors for the generation of new identifiers. Each translation rule usually execute a specific task, such as eliminating a certain variant of a construct (possibly introducing another construct), with most of the constructs left unchanged. Most of the rules that appear in our programs concern copy operations, while only few of them act as real translations. For example, the translation that performs step (3) would involve the rules for the following tasks:

3-i copy abstracts;

3-ii copy lexical attributes of abstracts;

3-iii replace relationships (only one-to-many and one-toone) with reference attributes;

3-iv copy generalizations.

As en example of Datalog implementation of rules, Fig. 2.2 shows rules (3-i), as an example of a copy rule; and (3-iii) as a translations rule, replacing binary one-to-many (or one-to-one) relationships with reference attributes.

Rule 3-i

```
1. ABSTRACT(
2.         OID: #abstract_0(absOid),
3.         Abs-Name: name,
4.         Schema: tgt )
5. ← ABSTRACT(
6.         OID: absOid,
7.         Abs-Name: name,
8.         Schema: src )
```

Rule 3-iii

```
1. ABSTRACTATTRIBUTE(
2.         OID: #abstractAttribute_1(aggrOid),
3.         Abstract: #abstract_0(absOid1),
4.         Att-Name: aggName,
5.         AbstractTo: #abstract_0(absOid2),
6.         IsOptional: isOpt1,
7.         Schema: tgt )
8. ← BINARYAGGREGATIONOFABSTRACTS(
9.          OID: aggrOid,
10.         Agg-Name: aggName,
11.         Abstract1: absOid1,
12.         IsOptional1: isOpt1,
13.         IsFunctional1: "TRUE",
14.         Abstract2: absOid2,
15.         Schema: src )
```

Figure 2.2: Basic Datalog rules.

Conditions for the applicability of the rule are placed in the body. Rule 3-i has no condition, and so it copies all abstracts. Differently, Rule 3-iii shows, in line 13, a condition that restrict its applicability only to aggregations that have `IsFunctional1=true`, representing one-to-many and one-to-one relationships. Row 2 of each rule generates a new construct instance in the dictionary with a new identifier generated by a Skolem function; Rule 3-i generates a new Abstract for each Abstract in the source schema (and it is a copy, except for the internal identifier), whereas Rule 3-iii generates a new AbstractAttribute for each BinaryAggregationOfAbstracts, with suitable features.

Since elementary translations refer to generic *supermodel* constructs, they can be easily reused, because each of them can be applied to all constructs that correspond to the same meta-construct. At the beginning of the translation process, rules for copying schemas from the specific source model to the *supermodel* are performed, and a final one for going back from the supermodel to the target model of interest.

Along with the schemas, it is possible to manage translations of actual data, derived from the translations of schemas. This is made possible by the use of

a dictionary for the data level, built in close correspondence with the schema level dictionary.

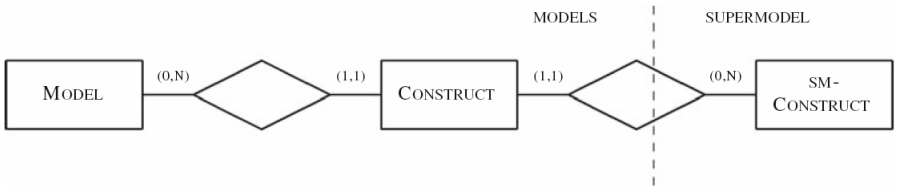


Figure 2.3: Relationships between models, constructs, meta-constructs

2.4 The relational dictionary

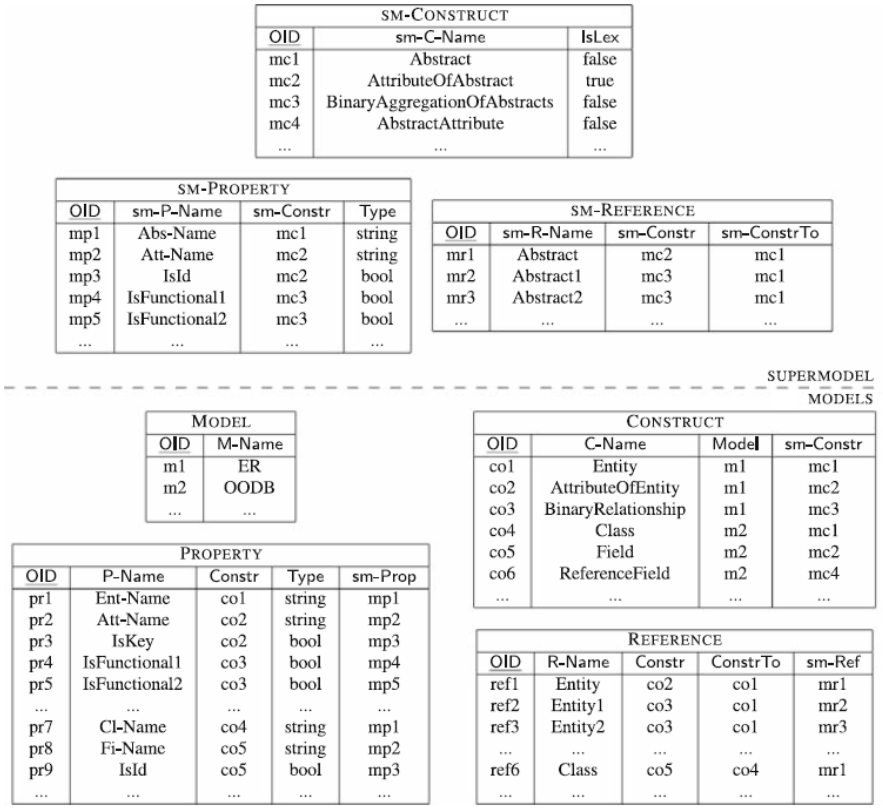
We have previously explained that one of the main concepts of our approach is the idea that a metamodel is a set of constructs (called metaconstructs) that can be used to define models, which are instances of the metamodel. For us a model is a set of elements called *constructs*, each of which with its own characteristics. Each construct correspond to a metaconstruct and the correspondence is one to one. Once we have defined the notions of model and metamodel, construct and metaconstruct, and their relationships, let us give some details about what a supermodel exactly represent. It is a model [1] that contains a construct for each metaconstruct, in the most general version. Therefore, each model can be seen as a specialization of the supermodel, except for renaming of constructs. In Fig. 2.3 the aforementioned concepts are represented in a diagram.

The expressivity of the supermodel can be enhanced as new metaconstructs and so sm-constructs can be added.

Let's now go deeper in some implementation details, to explain how we have concretely realized the overall approach. A portion of the relational dictionary is illustrated in Fig. 2.4, as we have implemented in our tool. The `SM-Construct` table contains the generic metaconstructs, in the picture are shown "Abstract," "AttributeOfAbstract," "BinaryAggregationOfAbstracts," "AbstractAttribute,".

The previous metaconstructs are used to classify each construct in the `Construct` table. Each row in this table, has a reference to a sm-construct

[1] Remember that we are always talking about models, just at different levels of abstraction



SM-CONSTRUCT

OID	sm-C-Name	IsLex
mc1	Abstract	false
mc2	AttributeOfAbstract	true
mc3	BinaryAggregationOfAbstracts	false
mc4	AbstractAttribute	false
...	...	...

SM-PROPERTY

OID	sm-P-Name	sm-Constr	Type
mp1	Abs-Name	mc1	string
mp2	Att-Name	mc2	string
mp3	IsId	mc2	bool
mp4	IsFunctional1	mc3	bool
mp5	IsFunctional2	mc3	bool
...	...	...	...

SM-REFERENCE

OID	sm-R-Name	sm-Constr	sm-ConstrTo
mr1	Abstract	mc2	mc1
mr2	Abstract1	mc3	mc1
mr3	Abstract2	mc3	mc1
...	...	...	...

SUPERMODEL

MODELS

MODEL

OID	M-Name
m1	ER
m2	OODB
...	...

CONSTRUCT

OID	C-Name	Model	sm-Constr
co1	Entity	m1	mc1
co2	AttributeOfEntity	m1	mc2
co3	BinaryRelationship	m1	mc3
co4	Class	m2	mc1
co5	Field	m2	mc2
co6	ReferenceField	m2	mc4
...	...	...	...

PROPERTY

OID	P-Name	Constr	Type	sm-Prop
pr1	Ent-Name	co1	string	mp1
pr2	Att-Name	co2	string	mp2
pr3	IsKey	co2	bool	mp3
pr4	IsFunctional1	co3	bool	mp4
pr5	IsFunctional2	co3	bool	mp5
...	...	...	...	...
pr7	Cl-Name	co4	string	mp1
pr8	Fi-Name	co5	string	mp2
pr9	IsId	co5	bool	mp3
...	...	...	...	...

REFERENCE

OID	R-Name	Constr	ConstrTo	sm-Ref
ref1	Entity	co2	co1	mr1
ref2	Entity1	co3	co1	mr2
ref3	Entity2	co3	co1	mr3
...	...	...	...	...
ref6	Class	co5	co4	mr1
...	...	...	...	...

Figure 2.4: A portion of the MIDST relational dictionary

(by means of the sm-Constr column) and to a model (by means of the Model column). For example the rows of the `Construct` table, with `C-Name` values *Entity* and *Class*, shows that the two constructs are related with two different models (namely *ER* and *OODB*) but with the same sm-construct *Abstract*.

Properties of each sm-construct are stored in the `SM-Property` table. This table contains records with the name of the property, the reference to the sm-construct and the type of the property. Going down at the underlying abstraction level, we find the corresponding constructs properties, represented by the `Property` table.

The table `SM-Reference` is used to manage the relationships between the various sm-constructs. In the picture we can see that the sm-construct "AttributeOfAbstract" is related to an "Abstract", at the first row. The "BinaryAggregationOfAbstracts" instead, is clearly exploited to define binary relationships between abstracts. Also in this case we have the model level representation. So an "AttributeOfEntity" is related to an "Entity", while a "BinaryRelationship" involve two entities.

To define restrictions on a model we have the possibility of specifying conditions on the properties for a construct. For example, to define an object model that does not allow the specification of identifying fields, we could add a condition that says that the property "IsId associated with "Field is identically "false. These restrictions can be expressed as propositional formulas over the properties of constructs.

SCHEMA

OID	S-Name
s1	SchemaER1
s2	SchemaER2

ENTITY

OID	Ent-Name	Schema
e1	School	s1
e2	Professor	s1
e3	Course	s1
e4	Employee	s2
e5	Project	s2

ATTRIBUTEOFENTITY

OID	Entity	Att-Name	Type	isKey	Schema
a1	e1	Code	int	true	s1
a2	e1	SName	string	false	s1
a3	e1	City	string	false	s1
a4	e2	SSN	int	true	s1
a5	e2	Name	string	false	s1
a6	e3	CN	int	true	s1
a7	e3	Title	string	false	s1
a8	e4	EN	int	true	s2
...	...	...	...	...	...

BINARYRELATIONSHIP

OID	Rel-Name	Entity1	IsOpt1	IsFunctional1	Entity2	...	Schema
b1	Membership	e2	false	true	e1	...	s1
b2	Teaching	e3	true	true	e2	...	s1
b3	Participation	e4	false	false	e5	...	s2

Figure 2.5: ER model dictionary

The two layers, Supermodel and Models, have a quite similar structure. The higher level represents the unique, generic supermodel, while at the lower level we can find the different models that we are able to describe. The supermodel's set of constructs is defined but can be extended and one extension, with the goal of enhancing the expressivity to the Semantic Web world, is one of the tasks of my research activity.

Model specific dictionaries have one table for each construct, with their respective properties and references. At this level we can specify the schemas

of interest. The binary ER model representation is depicted in Fig. 2.5, it includes tables `Entity`, `AttributeOfEntity`, and `BinaryRelationship`.

In these tables we have stored the information about the two schemas sketched in Fig. 2.6.

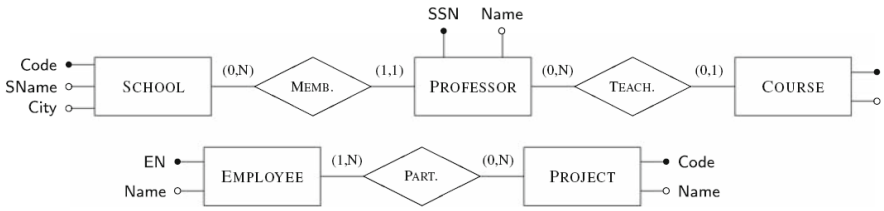


Figure 2.6: Two samples of ER schemas

Concluding, we can see the relational dictionary as composed of four parts, with two coordinates, as depicted in Fig. 2.6. We can distinguish schemas (lower portion) versus models (upper portion) and model-specific (left portion) versus supermodel (right portion).

	model specific	*model independent*
model descriptions (the "metalevel")	models	supermodel
schema descriptions	model-specific schemas	supermodel schemas

Figure 2.7: Partition of the MIDST abstraction layers

2.5 Conclusions

This chapter has the goal of introducing to MIDST, an implementation of the ModelGen operator that supports model-generic translation of schemas. The

translation approach we here presented, is characterized by the ability of "modeling models", at different levels of abstraction. At the higher level the most generic model, that we have defined as the supermodel, can be exploited to describe more specific models, that actually are the ones involved in the translation process. Schemas and data can be transformed from a source to a target model by executing Datalog programs. The choice of Datalog allows an independence from any engine. The model independence is reached by referring rules to the metaconstructs, instead of model specific. In the following of the document we will propose extensions of MIDST that supports the interoperability between Semantic Web formalism and to translate data and schemas from databases to ontologies and viceversa.

Chapter 3

Translation between Semantic Web annotation formalisms

The Semantic Web relies on semantic annotations which describe information in a machine readable form. Two popular formalisms that have been conceived for this aim are Resource Description Framework (RDF) [**?**] and Topic Maps [**?**]. The Resource Description Framework is a model developed by the W3C for representing information about resources in the World Wide Web. Topic Maps is a standard for knowledge integration developed by the ISO. The two specifications were developed in parallel during the late 1990s within their separate organizations for what initially seamed very different purposes. The results, however, turned out to have a lot in common and this stimulate the interest for their unification. While unification has to date not been possible (for a variety of technical and political reasons), a number of attempts have been made to uncover the similarities between RDF and Topic Maps and to find ways of achieving interoperability at the data level. There is a huge quantity of interesting and useful information represented both in RDF and Topic Maps and the trend is that the quantity of such information is increasing.

In this chapter, we first illustrate the RDF and Topic maps specification then we present an analysis of the existing approaches about the translations between RDF and Topic Maps. We then introduce the approach we choose to follow to address this issue.

3.1 RDF

The Resource Description Framework (RDF) is a language for representing information about resources in the World Wide Web. It is specifically used for representing metadata about Web resources, such as the title, author, and modification date of a Web page, copyright and licensing information about a Web document. However, by generalizing the concept of a "Web resource", RDF can also be used to represent information about things that can be identified on the Web, even when they cannot be directly retrieved on the Web. Examples include information about items available from on-line shopping facilities (e.g., information about specifications, prices, and availability), or the description of a Web user's preferences for information delivery.

RDF provides a common framework for expressing information so it can be exchanged between applications without loss of meaning. Since it is a common framework, application designers can leverage the availability of common RDF parsers and processing tools. The ability to exchange information between different applications means that the information may be made available to many more applications other than those for which it was originally created.

Basics

RDF is based on the idea of identifying things using Web identifiers (called Uniform Resource Identifiers, or URIs), and describing resources in terms of simple properties and property values. This enables RDF to represent simple statements about resources as a graph of nodes and arcs representing the resources, and their properties and values. In Figure 3.1 let us give a simple example of the RDF representation of the following statements "there is a Person identified by http://www.w3.org/People/EM/contact#me, whose name is Eric Miller, whose email address is em@w3.org, and whose title is Dr." :

This example, although being small, gives the idea of how RDF uses URIs to identify almost everything: individuals (Eric Miller), kinds of things (Person), properties (mailbox), values of those properties (mailto:em@w3.org).

Syntaxes

RDF also provides syntaxes for storing and exchanging these graphs, the most common are Notation-3 [**?**] and an XML-based syntax called RDF/XML . In the following there is a small chunk of RDF in RDF/XML corresponding to the graph in Figure 3.1:

Figure 3.1: RDF Example

```
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
             xmlns:contact="http://www.w3.org/2000/10/swap/pim/contact#">

  <contact:Person rdf:about="http://www.w3.org/People/EM/contact#me">
    <contact:fullName>Eric Miller</contact:fullName>
    <contact:mailbox rdf:resource="mailto:em@w3.org"/>
    <contact:personalTitle>Dr.</contact:personalTitle>
  </contact:Person>

</rdf:RDF>
```

Notice that this RDF document also contains URIs, as well as properties like `mailbox` and `fullName` (in an abbreviated form), and their respective values `em@w3.org`, and `Eric Miller`.

This RDF/XML document is machine processable and, using URIs, can link pieces of information across the Web. However, unlike conventional hypertext, RDF URIs can refer to any identifiable thing, including things that may not directly be findable on the Web (such as the person Eric Miller). The result

is that in addition to describing such things as Web pages, RDF can also describe cars, businesses, people, news events, etc. In addition, RDF properties themselves have URIs, to precisely identify the relationships that exist between the linked items.

Statements

RDF is based on the idea that the things being described have properties which have values, and that resources can be described by making statements, similar to those above, that specify those properties and values. RDF uses a particular terminology for talking about the various parts of statements. Specifically, the part that identifies the thing the statement is about is called the subject. The part that identifies the property or characteristic of the subject that the statement specifies (creator, creation-date, or language in these examples) is called the predicate, and the part that identifies the value of that property is called the object.

To make these kinds of statements suitable for processing by machines it is necessary to have a system of machine-processable identifiers for identifying a subject, predicate, or object in a statement and a machine-processable language for representing these statements and exchanging them between machines. Fortunately, the existing Web architecture provides both these necessary facilities.

Let's see how RDF uses URIs to make statements about resources. In RDF, the English statement: "http://www.example.org/index.html has a creator whose value is John Smith" could be represented by an RDF statement having:

- a subject http://www.example.org/index.html
- a predicate http://purl.org/dc/elements/1.1/creator
- and an object http://www.example.org/staffid/85740

Note how URIrefs are used to identify not only the subject of the original statement, but also the predicate and object, instead of using the words "creator" and "John Smith", respectively (some of the effects of using URIrefs in this way will be discussed later in this section).

Graphs and triples

RDF models statements can be expressed as nodes and arcs in a graph. In this notation, a statement is represented by a node for the subject, a node for

the object and an arc for the predicate, directed from the subject node to the object node. The subject must be a resource while the object can be either a resource or a literal. So the RDF statement above would be represented by the graph shown in Figure 3.2:

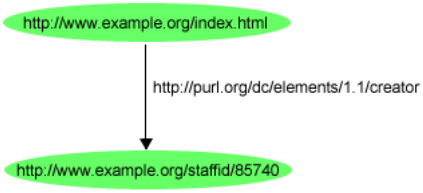


Figure 3.2: RDF Graph Example

An alternative way of writing down the statements, called triples, is also used, mostly exploited to store rdf data in documents. In the triples notation, each statement in the graph is written as a simple triple of subject, predicate, and object, in that order. For example, the three statements shown in Figure 3.3

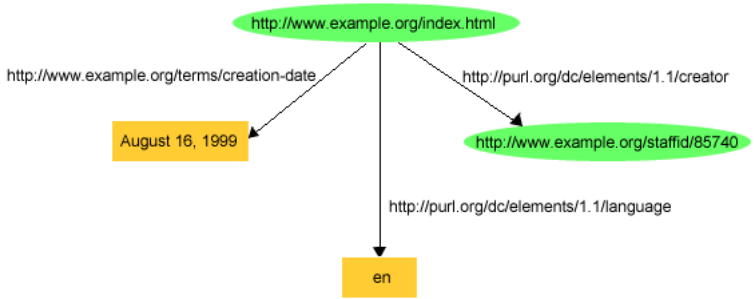


Figure 3.3: Statements in an RDF Graph representation

would be written in the triples notation as: b

```
<http://www.example.org/index.html>
        <http://purl.org/dc/elements/1.1/creator>
                <http://www.example.org/staffid/85740> .
```

```
<http://www.example.org/index.html>
        <http://www.example.org/terms/creation-date>
                "August 16, 1999" .

<http://www.example.org/index.html>
        <http://purl.org/dc/elements/1.1/language>
                "en" .
```

Each triple corresponds to an arc in the graph, with the arc's beginning and ending nodes (the subject and object of the statement). Unlike drawing the graph, the triples notation requires that a node be separately identified for each statement it appears in. So, for example, `http://www.example.org/index.html` appears three times (once in each triple) in the triples representation of the graph, but only once in the drawn graph. However, the triples represent exactly the same information as the drawn graph, and this is a key point: what is fundamental to RDF is the graph model of the statements. The notation used to represent or depict the graph is secondary.

The full triples notation requires that URI references be written completely, in angle brackets, which, as the example above illustrates, can result in very long lines on a page. Due to space limitations, triples elements here are reported on different lines but in the triple representation they actually should be presented in a single line.

Blank nodes

With its structure made of triples, RDF seams not able to represent a composite attribute. Let's consider the information concerning John's address "1501 Grant Avenue, Bedford, Massachusetts 01730" in a triple form:

```
exstaff:85740
        exterms:address
                "1501 Grant Avenue, Bedford, Massachusetts 01730" .
```

and we would like to have each element of the address separated. When it is necessary to store structured information like this in RDF, it is possible to consider the aggregate thing to be described (like John Smith's address) as a resource, and then making statements about that new resource. So, in the RDF graph, in order to separate John Smith's address into its components, a node is created to represent the concept of John Smith's address. RDF statements (additional arcs and nodes) can then be written with that node as the subject,

to represent the additional information, producing the graph shown in Figure 3.4:

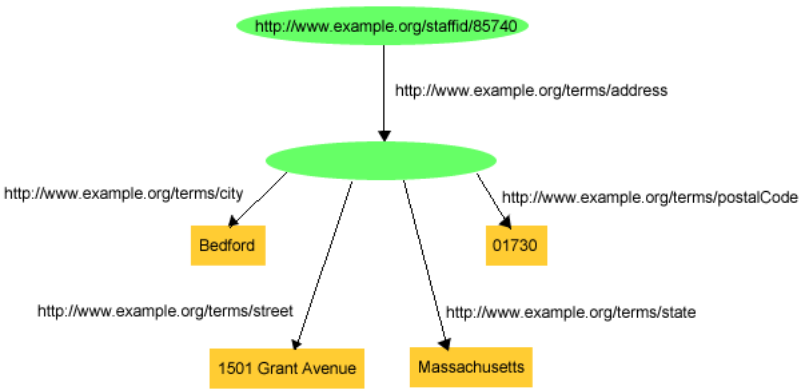


Figure 3.4: Structured information representation in RDF

In the graph, a node without a URIref is used to stand for the concept of "John Smith's address".

A node without an URI is commonly called *blank node*. This blank node serves its purpose in the drawing without needing a URIref, since the node itself provides the necessary connectivity between the various other parts of the graph. However, some form of explicit identifier for that node is needed in order to represent this graph as triples. To see this, it is enough to try to write the triples corresponding to what is shown in Figure 3.4, it would produce something like:

```
exstaff:85740   exterms:address   ??? .

??? exterms:street "1501 Grant Avenue" .

??? exterms:city "Bedford" .

??? exterms:state "Massachusetts" .

??? exterms:postalCode "01730" .
```

where ??? stands for something that indicates the presence of the blank node. Since a complex graph might contain more than one blank node, it is necessary to differentiate between them. As a result, triples use blank node identifiers (that are not universal but have a local scope), having the form

`_:name`, to indicate the presence of blank nodes. For instance, in this example a blank node identifier `_:johnaddress` might be used to refer to the blank node, in which case the resulting triples might be:

```
exstaff:85740   exterms:address         _:johnaddress .

_:johnaddress   exterms:street          "1501 Grant Avenue" .

_:johnaddress   exterms:city            "Bedford" .

_:johnaddress   exterms:state           "Massachusetts" .

_:johnaddress   exterms:postalCode      "01730" .
```

In a triples representation of a graph, each distinct blank node in the graph is given a different blank node identifier. Unlike URIrefs and literals, blank node identifiers are not considered to be actual parts of the RDF graph (this can be seen by looking at the drawn graph in Figure 6 and noting that the blank node has no blank node identifier). Blank node identifiers are just a way of representing the blank nodes in a graph (and distinguishing one blank node from another) when the graph is written in triple form.

Typing

A relevant feature of RDF that find an interesting application on blank nodes is the `rdf:type` property; we have already mentioned this in the examples, let us give a more detailed explanation. When an `rdf:type` property is used, the value of that property is considered to be a resource that represents a category or class of things, and the subject of that property is considered to be an instance of that category or class. It is common in RDF for resources to have `rdf:type` properties that describe the resources as instances of specific types or classes. Such resources are called *typed nodes* in the graph, or *typed node elements* in the RDF/XML. With respect to blank nodes this feature comes out particularly interesting. A blank node itself does not represent anything but it is representative by the properties it is connected with. This is helpful in representing real world things, since, for example, a person cannot be directly represented by a URI but can be described by a node of type person with properties name, surname, age, etc. etc.

Containers and collections

RDF provides a number of additional capabilities, such as built-in types and properties for representing groups of resources.

There is often a need to describe groups of things: for example, to say that a book was created by several authors, or to list the students in a course, or the software modules in a package. RDF provides several predefined (built-in) types and properties that can be used to describe such groups.

First, RDF provides a container vocabulary consisting of three predefined types (together with some associated predefined properties). A container is a resource that contains things. The contained things are called members. The members of a container may be resources (including blank nodes) or literals. RDF defines three types of containers: `rdf:Bag, rdf:Seq, rdf:Alt`

A Bag (a resource having type `rdf:Bag`) represents a group of resources or literals, possibly including duplicate members, where there is no significance in the order of the members. For example, a Bag might be used to describe a group of part numbers in which the order of entry or processing of the part numbers does not matter.

A Sequence or Seq (a resource having type `rdf:Seq`) represents a group of resources or literals, possibly including duplicate members, where the order of the members is significant. For example, a Sequence might be used to describe a group that must be maintained in alphabetical order.

An Alternative or Alt (a resource having type `rdf:Alt`) represents a group of resources or literals that are alternatives (typically for a single value of a property). For example, an Alt might be used to describe alternative language translations for the title of a book, or to describe a list of alternative Internet sites at which a resource might be found. An application using a property whose value is an Alt container should be aware that it can choose any one of the members of the group as appropriate.

To describe a resource as being one of these types of containers, the resource is given an rdf:type property whose value is one of the predefined resources rdf:Bag, rdf:Seq, or rdf:Alt (whichever is appropriate). The container resource (which may either be a blank node or a resource with a URIref) denotes the group as a whole. The members of the container can be described by defining a container membership property for each member with the container resource as its subject and the member as its object. These container membership properties have names of the form rdf:_n, where n is a decimal integer greater than zero, with no leading zeros, e.g., rdf:_1, rdf:_2, rdf:_3, and so on, and are used specifically for describing the members of containers. Container resources may also have other properties that describe the container, in addition to the container membership properties and the rdf:type property.

It is important to understand that while these types of containers are described using predefined RDF types and properties, any special meanings as-

sociated with these containers, e.g., that the members of an Alt container are alternative values, are only intended meanings.

A typical use of a container is to indicate that the value of a property is a group of things. For example, to represent the sentence "Course 6.001 has the students John, Lilian, Mark, Maria, and Phillip", the course could be described by giving it a `s:students` property (from an appropriate vocabulary) whose value is a container of type `rdf:Bag` (representing the group of students). Then, using the container membership properties, individual students could be identified as being members of that group.

Since the value of the `s:students` property in this example is described as a Bag, there is no intended significance in the order given for the URIrefs of the students, even though the membership properties in the graph have integers in their names. It is up to applications creating and processing graphs that include rdf:Bag containers to ignore any (apparent) order in the names of the membership properties.

RDF/XML provides some special syntax and abbreviations to make it simpler to describe such containers as shown in the next example:

```
<?xml version="1.0"?>
<rdf:RDF
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
         xmlns:s="http://example.org/students/vocab#">

   <rdf:Description rdf:about="http://example.org/courses/6.001">
     <s:students>
       <rdf:Bag>
         <rdf:li rdf:resource="http://example.org/students/Amy"/>
         <rdf:li rdf:resource="http://example.org/students/Mohamed"/>
         <rdf:li rdf:resource="http://example.org/students/Johann"/>
         <rdf:li rdf:resource="http://example.org/students/Maria"/>
         <rdf:li rdf:resource="http://example.org/students/Phuong"/>
       </rdf:Bag>
    </s:students>
   </rdf:Description>
</rdf:RDF>
```

A limitation of the containers is that there is no way to close them, i.e., to say "these are all the members of the container". A container only says that certain identified resources are members; it does not say that other members

do not exist. Also, while one graph may describe some of the members, there is no way to exclude the possibility that there is another graph somewhere that describes additional members. RDF provides support for describing groups containing only the specified members, in the form of RDF collections. An RDF collection is a group of things represented as a list structure in the RDF graph. This list structure is constructed using a predefined collection vocabulary consisting of the predefined type `rdf:List`, the predefined properties `rdf:first` and `rdf:rest`, and the predefined resource `rdf:nil`. The graph in Figure 3.5 illustrate the usage of these constructs.

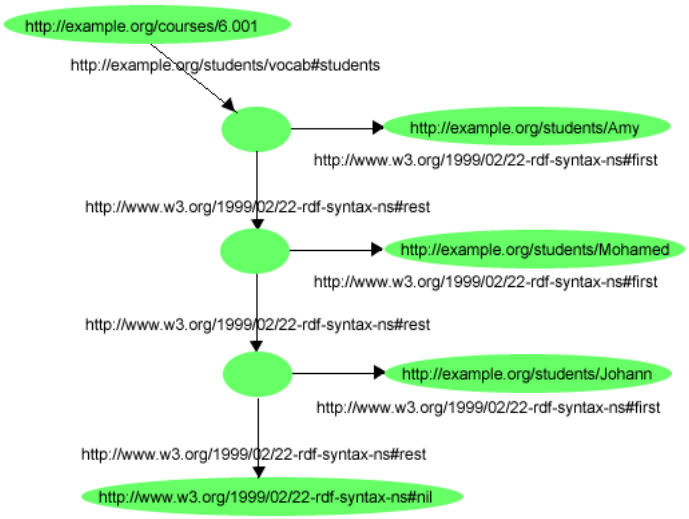


Figure 3.5: Example of an RDF collection

As already mentioned, RDF imposes no well-formed conditions on the use of the collection vocabulary so, when writing triples, it is possible to define RDF graphs with structures other than the well-structured graphs that would be automatically generated by using rdf:parseType="Collection". For example, it is not illegal to assert that a given node has two distinct values of the rdf:first property, to create structures that have forked or non-list tails, or to simply omit part of the description of a collection. Also, graphs defined by using the collection vocabulary in longhand could use URIrefs to identify the components of the list instead of blank nodes unique to the list structure. In this case,

it would be possible to create triples in other graphs that effectively added elements to the collection, making it non-closed.

As discussed in the preceding sections, RDF is intended to be used to express statements about resources in the form of a graph, using specific vocabularies (names of resources, properties, classes, etc.). RDF is also intended to be the foundation for more advanced languages, such as RDF Schema and OWL.

To query RDF documents many languages have been developed but one of the most popular is SPARQL. SPARQL can be used to express queries across diverse data sources, whether the data is stored natively as RDF or viewed as RDF via middleware. SPARQL contains capabilities for querying required and optional graph patterns along with their conjunctions and disjunctions. SPARQL also supports extensible value testing and constraining queries by source RDF graph. The results of SPARQL queries can be results sets or RDF graphs.

3.2 Topic Maps

Topic Maps provides an approach that unify the best of several worlds, including those of traditional indexing, library science and knowledge representation, with advanced techniques of linking and addressing.

Topic maps were originally developed in the late 1990's as a way to represent back-of-the-book index structures so that multiple indexes from different sources could be merged. However, the developers quickly realized that they could create a model with potentially far wider application. The result of that work was published in 1999 as ISO/IEC 13250-Topic Navigation Maps.

In addition to describing the basic model of topic maps and the requirements for a topic map processor, the first edition of ISO 13250 included an interchange syntax based on SGML and the hypermedia linking language known as HyTime. The second edition, published in 2002, added an interchange syntax based on XML and XLink. This is the syntax with the widest support in topic map processing products, and is the syntax that we adopt in this document.

Basis

The Topic Maps paradigm describes a way in which complex relationships between abstract concepts and real-world resources can be described and interchanged using a standard XML syntax.

Topic

The model of Topic Maps is (intuitively) centered on the concept of *topic*. A topic, in its most generic sense, can be anything whatsoever a person, an entity, a concept, really anything regardless of whether it exists or has any other specific characteristics, about which anything whatsoever may be asserted by any means whatsoever.

Only a non-definition can be more general than this definition.

The topic stands for a subject, the term used for the real world *thing* that the topic itself stands in for. Trying to go deeper in philosophy, the *subject* corresponds to what Plato called an idea. A topic, on the other hand, is like the shadow that the idea casts on the wall of Plato's cave: It is an object within a topic map that represents a subject. Strictly speaking, the term `topic` refers to the object or node in the topic map that represents the subject being referred to. However, there is (or should be) a one-to-one relationship between topics and subjects, with every topic representing a single subject and every subject being represented by just one topic. Then the two terms can, almost always, be used interchangeably.

So, in the context of a dictionary of opera, topics might represent subjects such as *Tosca*, *Madame Butterfly*, *Rome*, *Italy*, the composer *Giacomo Puccini*, or his birthplace, *Lucca*: that is, anything that might have an entry in the dictionary.

Topics can be categorized according to their kind. In a topic map, any given topic is an instance of zero or more topic types. This corresponds to the categorization inherent in the use of multiple indexes in a book (index of names, index of works, index of places, etc.), and to the use of typographic and other conventions to distinguish different types of topics.

Thus, Puccini would be a topic of type *composer*, Tosca and Madame Butterfly topics of type *opera*, Rome and Lucca topics of type *city*, Italy a topic of type *country*, etc. In other words, the relationship between a topic and its type is a typical class-instance relationship.

Exactly what one chooses to regard as topics in any particular application will vary according to the needs of the application, the nature of the information, and the uses to which the topic map will be put: In a thesaurus, topics would represent terms, meanings, and domains; in software documentation they might be functions, variables, objects, and methods; in legal publishing, laws, cases, courts, concepts, and commentators; in technical documentation, components, suppliers, procedures, error conditions, etc.

Topic types are themselves defined as topics by the standard. You must

explicitly declare *composer*, *opera*, *city*, etc. as topics in your topic map if you want to use them as types (in which case you will be able to say more about them using the topic map model itself).

Topics have three kinds of characteristics: names, occurrences, and roles in associations.

Names

Normally topics have explicit names, since that makes them easier to talk about. However, topics don't always have names: A simple cross reference, such as *see page 97*, is considered to be a link to a topic that has no (explicit) name.

Names exist in all the possible forms: as formal names, symbolic names, nicknames, pet names, everyday names, login names, etc. The topic map recognizes the need for some forms of name (that have particularly important and universally understood semantics) to be defined in a standardized way, in order for applications to be able to do something meaningful with them, and at the same time the need for complete freedom and extensibility to be able to define application-specific name types.

The standard therefore provides the facility to assign multiple base names to a single topic, and to provide variants of each base name for use in specific processing contexts.

The ability to be able to specify more than one topic name can be used to indicate the applicability of different names in different contexts or scopes (explained later), such as language, style, domain, geographical area, historical period, etc. A corollary of this feature is the topic naming constraint, which states that no two subjects can have exactly the same base name in the same scope.

Occurrences, as we have already seen, may be of any number of different types (we gave the examples of *monograph*, *article*, *illustration*, *mention* and *commentary* above). Such distinctions are supported in the standard by the concepts of occurrence role and occurrence role type.

Occurrences

A topic may be linked to one or more information resources that are deemed to be relevant to the topic in some way. Such resources are called occurrences of the topic.

An occurrence could be a monograph devoted to a particular topic, for example, or an article about the topic in an encyclopaedia; it could be a picture or video depicting the topic, a simple mention of the topic in the context of something else, a commentary on the topic (if the topic were a law, say), or any of a host of other forms in which an information resource might have some relevance to the subject in question.

Such occurrences are generally external to the topic map document itself (although they may also be inside it), and they are *pointed at* using whatever mechanisms the system supports, typically URIs (in XTM) or HyTime addressing (in HyTM). Today, most systems for creating hand-crafted indexes (as opposed to full text indexes) use some form of embedded markup in the document to be indexed. One of the advantages to using topic maps, is that the documents themselves do not have to be touched.

An important point to note here is the separation into two layers of the topics and their occurrences.

The distinction between an occurrence role and its type is subtle but important (at least in HyTM). In general terms they are both about the same thing, namely the way in which the occurrence contributes information to the subject in question (e.g. through being a portrait, an example or a definition). However, the role (indicated syntactically in HyTM by the role attribute) is simply a mnemonic; the type (indicated syntactically by the type attribute), on the other hand, is a reference to a topic which further characterizes the nature of the occurrence's relevance to its subject. In general it makes sense to specify the type of the occurrence role, since then the power of topic maps can be used to convey more information about the relevance of the occurrence.

Associations

Up to now, all the constructs that have been discussed have had to do with topics as the basic organizing principle for information. The concepts of *topic*, *topic type*, *name*, emphoccurrence and *occurrence role* allow us to organize our information resources according to topic (or subject), and to create simple indexes.

The really interesting thing, however, is to be able to describe relationships between topics, and for this the topic map standard provides a construct called the topic association.

A topic association asserts a relationship between two or more topics. Examples might be as follows:

Tosca was written by Puccini

Tosca takes place in Rome
Puccini was born in Lucca
Lucca is in Italy
Puccini was influenced by Verdi

Just as topics and occurrences can be grouped according to type (e.g., composer/opera/country and mention/article/commentary, respectively), so too can associations between topics be grouped according to their type. The association type for the relationships mentioned above are `written_by`, `takes_place_in`, `born_in`, `is_in` (or geographical containment), and `influenced_by`. As with most other constructs in the topic map standard, association types are themselves defined in terms of topics.

The ability to do typing of topic associations greatly increases the expressive power of the topic map, making it possible to group together the set of topics that have the same relationship to any given topic. This is of great importance in providing intuitive and user-friendly interfaces for navigating large pools of information.

It should be noted that topic types are regarded as a special (i.e. syntactically privileged) kind of association type; the semantics of a topic having a type (for example, of Tosca being an opera) could equally well be expressed through an association (of type emphtype-instance) between the topic *opera* and the topic emphTosca. The reason for having a special construct for this kind of association is the same as the reason for having special constructs for certain kinds of names (indeed, for having a special construct for names at all): The semantics are so general and universal that it is useful to standardize them in order to maximize interoperability between systems that support topic maps.

It is also important to note that while both topic associations and normal cross references are hyperlinks, they are very different creatures: In a cross reference, the anchors (or end points) of the hyperlink occur within the information resources (although the link itself might be outside them); with topic associations, we are talking about links (between topics) that are completely independent of whatever information resources may or may not exist or be considered as occurrences of those topics.

This is important because it means that topic maps are information structure that exist independently if actually connected to any information resources or not. The knowledge that Rome is in Italy, that Tosca was written by Puccini and is set in Rome, etc. etc. is useful and valuable, whether or not we have information resources that actually pertain to any of these topics.

Also, because of the separation between the information resources and the topic map, the same topic map can be overlaid on different pools of information,

just as different topic maps can be overlaid on the same pool of information to provide different *views* to different users. Furthermore, this separation provides the potential to be able to interchange topic maps among publishers and to merge one or more topic maps.

Each topic that participates in an association plays a role in that association called the association role. In the case of the relationship *Puccini was born in Lucca*, expressed by the association between Puccini and Lucca, those roles might be emphperson and *place*; for *Tosca was composed by Puccini* they might be *opera* and *composer*. It will come as no surprise now to learn that association roles can also be typed and that the type of an association role is also a topic!

Unlike relations in mathematics, associations are inherently multidirectional. In topic maps it doesn't make sense to say that A is related to B but that B isn't related to A: If A is related to B, then B must, by definition, be related to A. Given this fact, the notion of association roles assumes even greater importance. It is not enough to know that Puccini and Verdi participate in an emphinfluenced-by association; we need to know who was influenced by whom, i.e. who played the role of *influencer* and who played the role of *influencee*.

his is another way of warning against believing that the names assigned to association types (such as *was influenced by*) imply any kind of directionality. They do not! This particular association type could equally well (under the appropriate circumstances) be characterized by the name *influenced* (as in emphVerdi influenced Puccini).

Subject identity

The goal with topic maps is to achieve a one-to-one relationship between topics and the subjects that they represent, in order to ensure that all knowledge about a particular subject can be accessed via a single topic. However, sometimes the same subject is represented by more than one topic, especially when two topic maps are being merged. In such a situation it is necessary to have some way of establishing the identity between seemingly disparate topics. For example, if reference works publishers from Norway, France and Germany were to merge their topic maps, there would be a need to be able to assert that the topics *Italia*, *l'Italie* and emphItalien all refer to the same subject.

The concept that enables this is that of subject identity. When the subject is an addressable information resource (an *addressable subject*), its identity may be established directly through its address. However most subjects, such as Puccini, Italy, or the concept of opera, are not directly addressable. This

problem is solved through the use of subject indicators (originally called emph-subject descriptors in ISO 13250). A subject indicator is "a resource that is intended ... to provide a positive, unambiguous indication of the identity of a subject." Because it is a resource, a subject indicator has an address (usually a URI) that can be used as a *subject identifier*.

Any two topics that share one or more subject indicators (or that have the same subject address, in the case of addressable subjects) are considered to be semantically equivalent to a single topic that has the union of the characteristics (the names, occurrences and associations) of both topics. In a processed topic map a single topic node results from combining the characteristics of the two topics.

A subject indicator could be an official, publicly available document (for example, the ISO standard that defines 2- and 3-letter country codes), or it could simply be a definitional description within (or outside) one of the topic maps. A published subject indicator is a subject indicator that is published and maintained at an advertised address for the purpose of facilitating knowledge interchange and mergeability, either through topic maps or by other means.

Published subjects are a necessary precondition for the widespread use of portable topic maps, since there is no point in offering a topic map to others if it is not guaranteed to match with relevant occurrences in the receiver's pool of information resources. Activities are therefore underway, under the aegis of OASIS and others to develop recommendations for the documentation and use of published subjects.

Scope

The topic map model allows three things to be said about any particular topic: What names it has, what associations it participate in, and what its occurrences are. These three kinds of assertions are known collectively as topic characteristics.

Assignments of topic characteristics are always made within a specific context, which may or may not be explicit. For example, if we mention *tosca*, we should expect my readers to think of the opera by Puccini (or its principle character), because of the context that has been set by the examples used so far in this thesis. For an audience of bakers, however, the name *tosca* has quite other and sweeter connotations: it denotes another topic altogether.

Given two such simple statements as Tosca takes place in Rome and Tosca kills Scarpia, most of today's computers would not be able to infer which of the topics named *Tosca* was involved. In order to avoid this kind of problem,

topic maps consider any assignment of a characteristic to a topic, be it a name, an occurrence or a role, to be valid within certain limits, which may or may not be specified explicitly. The limit of validity of such an assignment is called its scope.

Scope is defined in terms of themes, and a theme is defined as a member of the set of topics used to specify a scope. In other words, a theme is a topic that is used to limit the validity of a set of assignments. The use of scope in topic maps can also aid navigation, for example by dynamically altering the view on a topic map based on the user profile and the way in which the map is used.

3.3 Models comparison

The two formalisms we have chosen to consider in this thesis are RDF and Topic Maps since they are commonly used in the Semantic Web context and a number of significative works, approaching the interoperability between, can be found in literature [**?**]. In this section we will give a brief description of both models, underlying which are the main differences and the matching points. Topic Maps and RDF have been conceived as a means to describe information resources with some different perspectives. Topic Maps are used to support high-level indexing to let information be easily findable for humans while RDF provides structured metadata and a foundation for logical inferencing to allow machines to interpret information resources. In Topic Maps the main construct is the Topic that is used to describe the subjects the map is about. In RDF there is an equivalent of the subject that is the resource represented by a node, both terms are very general and can be everything. The main differences between the two models arise when it is necessary to say something about the resources we are describing. RDF has just one, simple way to make assertions on resources, by statements on the form of a triple $< subject, predicate, object >$, where *subject* is the resource of interest, *predicate* is a resource denoting the property we are specifying and *object* represents the value of the property and can be a resource or a literal. Due to the statement structure, an RDF document can then be seen as a directed labeled graph. In Topic Maps there are mainly three ways to assert something about subjects, namely *associations*, *occurrences* and *names*. An association has a type and represents an n-ary, undirected relationship between topics where each participant plays a specific role. Since all can be described in a Topic Map is a *topic*, the association type and the participant role are topics themselves. Another relevant difference between the two formalisms

can be found in the way resources can be addressed and consequently in the way two elements can be considered identical. In RDF there can be three kinds of nodes: *literal*, *URI nodes* and *blank nodes*. Literals are constant values in the form of a string and the identity is established looking at the values. URI nodes are resources identified by a URI so the same URI represents the same thing. A blank node is an anonymous resource without URI. Considering that RDF allows only binary properties the common use of the blank node is to associate an URI node with a complex data (e.g. the address of a person composed by street, city, postal code, etc.), the analogous of a multivalue attribute in the ER model. To check if two blank nodes are the same, since they have no URI, it is necessary to evaluate the statements defined on them. There can be a doubt regarding identification of URI nodes: if the URI points to a document, the RDF node represents the document itself or the object the document is about. Topic Maps provide a mechanism to represent the two different possibilities. A subject can be addressed by a topic with a subject address, which means that the subject is identified by the resource or by a subject indicator that means the subject is identified by what the resource is about.

In the next section, we will see which are the main existing approaches to implement the interoperability between these two formalisms.

3.4 Related works

Evaluation criteria

We examined five existing proposals that we considered the most significative. They have been chosen as being sufficiently well documented to be suitable for a detailed examination. Each translation proposal has been evaluated against the following general criteria:

- Completeness: represents how each proposal is able to handle every semantic construct that can be expressed in the source model and provide a means to represent it without loss of information in the target model.
- Naturalness: expresses the degree to which the results of a translation correspond to the way in which someone familiar with the target paradigm would naturally express the information content. Naturalness is strictly related to the readability of the result.

Naturalness is extremely important because the result of an unnatural translation is structurally different from data that was originally created in the target

model. An uncomplete representation is evaluated with respect to the capacity of the approach to be extended with features that complete its expressivity.

Types of mappings

All the existing approaches fall into two distinct categories that in [**?**] are called *object mappings* and *semantic mappings*. The two approaches can be summed up as follows:

- Object mappings use the low-level building blocks of one language to describe the object model of the other. For example, assuming for now that the structure of a simple binary association is a quintuple, consisting of one (a)ssociation, two (r)ole types, and two role (p)layers (p-r-a-r-p), that association would in an object mapping to RDF be represented as four statements that relate five resources.

- Semantic mappings start from higher level concepts that carry the semantics of each model and attempt to find equivalences between them. Abinary association in Topic Maps would be seen to represent the same kind of thing that is often represented by an RDF statement (i.e., a relationship between two entities) and would therefore be represented using a single RDF statement. Where no direct semantic equivalent can be found, the missing semantics are defined using the facilities available in one of the two paradigms, i.e., classes, properties, or published subjects.

The advantage of an object mapping is that it is easy to make it generic (provided, of course, that the object model on which it is based is complete) and this ensures completeness without any additional effort. The disadvantage is the unnaturalness of the result. Semantic mappings yield much more natural results but suffer from the disadvantage that generality is much harder to ensure and may in some cases require additional information not always present in the source document.

Existing proposals

Five existing proposals are here analyzed. They will be referred to by the names of their authors or, in the case of multiple authors, by the name of the organization to which the authors are affiliated. Each proposal is characterized here in terms of the translation directions that cover: i.e., RDF to Topic Maps

(RDF2TM), and Topic Maps to RDF (TM2RDF), respectively. They are, in chronological order:

Moore RDF2TM and TM2RDF proposal described in [**?**]. Not implemented.

Stanford TM2RDF proposal described in [**?**]. Implemented.

Ogievetsky TM2RDF proposal described in [**?**]. Implemented in the XTM2RDF Translator.

Garshol RDF2TM and TM2RDF proposal described in [**?**] and [**?**]. Documented in [**?**], [**?**], and [**?**], and implemented in Ontopia Knowledge Suite.

Unibo RDF2TM and TM2RDF proposal described in [**?**] and [**?**]. Implemented in Meta.

Two simple test cases are used to enable an evaluation of the criterion naturalness. These test cases are not intended to be complete since their purpose is just to give an intuition for the kind of results produced by the various proposals.

Both test cases are separated into instance data (above the dotted line comment) and ontology or schema data that might normally be expected to come from a shared document, such as a topic map ontology or the RDF namespace document respectively.

TM2RDF test case

```
 [puccini : person   = "Giacomo Puccini"]
[tosca   : opera    = "Tosca"]

{tosca, premiere-date, [[1900-01-14]]} {tosca, synopsis,
"http://www.azopera.com/learn/synopsis/tosca.shtml"}

composed-by( tosca : work, puccini : composer )

              /* ----------------------------------- */

[person        = "Person"
@"http://psi.ontopia.net/music/#person"]
[composer      = "Composer"
```

```
@"http://psi.ontopia.net/music/#composer"]
[opera          = "Opera"
@"http://psi.ontopia.net/music/#opera"]
[work           = "Work"
@"http://psi.ontopia.net/music/#work"]

[premiere-date = "Premire date"
@"http://psi.ontopia.net/music/#premiere-date"]
[synopsis       =
"Synopsis"       @"http://psi.ontopia.net/music/#synopsis"]
[composed-by    = "Composed by"
@"http://psi.ontopia.net/music/#composed-by"]
```

RDF2TM test case

```
@prefix music: <http://psi.ontopia.net/music/#> .
@prefix rdf:
<http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs:
<http://www.w3.org/2000/01/rdf-schema#> .

[ rdf:type music:opera;
  rdfs:label "Tosca";
  music:premiere-date "1900-01-14";
  music:synopsis
  <http://www.azopera.com/learn/synopsis/tosca.shtml>;
  music:composed-by [
    rdf:type music:person;
    rdfs:label "Giacomo Puccini" ]
] .

               # ---------------------------------------

music:person
rdfs:label "Person" .
music:opera
rdfs:label "Opera" .
```

```
music:composed-by rdfs:label "Composed by" . \\music:premiere-date
rdfs:label "Premire date" . \\music:synopsis rdfs:label "Synopsis"
.
```

Moore

[**?**] was the first paper to address the issue of interoperability between RDF and Topic Maps. The paper starts out by presenting data models developed by the author that "capture the isness [sic] of the two paradigms". Having presented the two models, Moore introduces the distinction between what he calls "mapping the model" and "modelling the model". The key difference is that the first is "semantic", whereas the second "uses each standard as a tool for describing other models". This become the basis for the definition of "semantic mapping" and "object mapping", respectively, that we have already introduced.

Moore provides examples of both strategies but express a clear preference for the semantic mapping. The reason for this is that a goal is to be able to run, say, a TMQL query against an RDF model and get expected results. Moore points out that this is only possible when a semantic mapping approach is used.

Moore's RDF2TM object mapping approach is based on defining PSIs for every RDF construct in his model (i.e., resource, statement, property, subject, object, identity, literal, and model) and expressing RDF statements as ternary associations of type `rdf-statement` using the role types `rdf-subject`, `rdf-property` and `rdf-object`. This raises issues with the handling of literals (since role players in associations cannot be strings) to which no solution is proposed.

The TM2RDF object mapping approach is based on defining RDF properties for each TM construct as follows: topic, `topicassoc, instanceof, topicassocmember, roleplayingtopic, roledefiningtopic, topicoccur, topicname, topicnamevalue, scopeset, subjindicatorref, resourceref`. An example of a simple binary association is given that involves five topics (for the association type and role types, in addition to the role-playing topics). The RDF equivalent of this requires 22 statements, three for each of the five topics, and seven for the association itself.

Moore concludes that the object mapping approach, while interesting, is of limited usefulness, and he goes on to describe a semantic mapping approach (which he calls "mapping the model") based on the observation that RDF is concerned with describing the arcs between entities with identity [whereas]

Topic Maps is concerned with describing typed relationships between entities with identity. A number of semantic equivalences are defined, as follows:

Table 3.1: Moore RDF-Topic Maps mapping

RDF	Topic Maps
RDF model	Topic Map
Identity	SubjectIndicatorReference
Resource	Topic
Statement	Association (approximate)

The mapping from RDF statement to association is identified as being problematic because "RDF has three pieces of information and Topic Map associations have five", leading the author to suspect that a "complete" semantic mapping of the models may not be possible. The remainder of the paper is devoted to examining how to represent RDF statements as associations and vice versa.

RDF statements are viewed as binary associations whose role-players correspond to the subject and object of the statement and have the role types 'subject' and 'object' respectively. The mechanism for representing the property of the statement is not fully defined, since the text and the diagram contradict each another. However, both text and diagram assign some significance to the name of the topic that represents the subject role.

According to Moore, this approach has a problem in that 'arc' is "not a first class entity in the TopicMap model". Why this should be a problem is not made clear, but Moore advocates solving it by extending the Topic Maps model with the notion of arcs (and association templates).

A different approach is employed in order to view associations as RDF statements. An incomplete example shows a binary association represented as two RDF statements, with the role-playing topics being the subject and object in the one and the object and subject in the other. This approach is perhaps motivated by the recognition that RDF statements have direction whereas associations do not. However this is not stated explicitly; nor is it clear how the approach would work with associations that involve more than two role players.

Moore's object mapping approach is reasonably complete, whereas his semantic mapping approach is just a sketch that focuses on RDF statements and associations. Other constructs like names, occurrences and scope are not

covered. Neither approach is reversible. In the case of the object mapping approach, the assumption is that one is working in one domain or the other, but not in both. In the case of the semantic mapping approach, the fact that a statement maps to a single association whereas an association maps to two statements shows that translations cannot be reversed.

Semantic mappings are shown to be superior to object mappings in terms of naturalness. The latter yields unnatural results in both directions. Whatever the direction, a "natural" source document leads to an "unnatural" result and achieving a "natural" result is only possible if the starting point is "unnatural". In the object mapping example given in the [**?**], a simple binary association translates to 22 RDF statements.

Moore's semantic mapping approach, on the other hand, achieves a more natural result: Going from Topic Maps to RDF, a binary association requires two RDF statements; going the other way, an RDF statement maps to a single association.

Standford

Lacher and Decker [**?**] focus on making it possible to query Topic Maps using an "RDF-aware infrastructure" that was co-developed by one of the authors. This proposal is thus TM2RDF only.

Reference is made to the layered integration model of data interoperability which separates the data integration problem into three quasi-independent layers: the syntax layer, the object layer, and the semantic layer. The idea is to build an RDF representation of the topic map on the object layer and then perform a "bijective graph transformation" such that the topic map can be viewed as RDF. Ignoring the syntax layer means that the approach will work with both the SGML and the XML serialization syntaxes of Topic Maps. Ignoring the semantic layer (i.e., adopting what we have termed an object mapping approach) has the advantage, according to the authors, that all information is preserved. (The authors point out that a semantic mapping "could possibly lead to a loss of information".)

Instead of defining their own model for Topic Maps, Lacher and Decker use PMTM4, the Processing Model for Topic Maps, proposed by Newcomb and Biezunski in [PMTM4], which has since been superseded by [TMDM].

PMTM4 is a graph model consisting of three node types (for topics, associations, and scopes), and four arc types: associationMember (aM), associationScope (aS), associationTemplate (aT), and scopeComponent (sC). The aM arc is "peculiar" in that it is both typed and labelled (and thus effectively has

three ends) in order to connect the association with both the role-player and its role (or role type). Names and occurrences are regarded as specializations of associations; URIs and strings are not part of the model.

To illustrate their approach Lacher and Decker show a simple (untyped) association between the country Denmark (which has a name) and the natural resource petroleum. This is represented as a PMTM4 graph consisting of eight t-nodes, two a-nodes, four aM arcs, and one aT arc. The (binary) association between Denmark and petroleum requires two aM arcs (one for each role), and so does the name "Denmark" (since topic names are regarded in PMTM4 as a kind of binary association).

Lacher and Decker define RDF classes and properties for each of the PMTM4 node types and arc types. The transformation consists essentially of replacing a-, t-, and s-nodes with RDF nodes of corresponding types, and replacing arcs with corresponding properties. However in order to handle the "three-legged" aM arcs, reification is necessary, thus introducing one new RDF node and four new properties (rdf:subject, rdf:predicate, rdf:object and tms:roleLabel) for each aM arc. The resulting "RDF Topic Map graph" is shown as a figure consisting of a total of 17 nodes and 20 arcs.

The authors opt to represent each undirected PMTM4 arc by a single, directed RDF arc (rather than two arcs) in order to avoid consistency problems, pointing out that while this is not a lossy transformation, it does require consideration when formulating queries.

No syntax example is given in [**?**] to show the result of the transformation but from the text it is clear that node identity is either based on source locators (where XML IDs were specified in the source topic map) or else generated (where no IDs were specified). Subject identifiers and subject locators are not used presumably because the PMTM4 model does not extend to identifiers.

The Stanford approach is complete with respect to PMTM4, but the latter is not a complete model for Topic Maps (since is does not handle URIs and strings). The Stanford proposal itself is therefore not complete.

The proposal does not score well in terms of naturalness since it requires upwards of 20 statements to represent information that would naturally be modelled using two statements in RDF

Ogievetsky

[**?**] describes both a method for transforming topic maps expressed in XTM syntax ([XTM1.0]) to RDF and the author's XSLT-based implementation of this approach in the XTM2RDF Translator. Transformations are described in

terms of the processing of XTM elements and the approach is thus very syntax-oriented. The resulting RDF conforms to a vocabulary (called RTM) which consists of 11 classes and 17 properties defined partly in terms of XTM itself and partly in terms of PMTM4, the "processing model" proposed by Newcomb and Biezunski and described in the preceding section.

The classes and properties defined by the RTM vocabulary are:

rdfs:Class t-node, topic, scope, member, association, basename, variantname, occurrence, class-subclass, class-instance, templaterpc

rdf:Property association-role, validIn, indicatedBy, constitutedBy, name, templatedBy, role-topic, role-basename, role-variantname, role-occurrence, role-superclass, role-subclass, role-class, role-instance, role-template, role-role, role-rpc

Each `<topic>` element results in the creation of an RDF statement of type `rtm:topic`. The topic's subject locator (if any) becomes the URI of the subject of the statement, otherwise a blank node is created. Subject identifiers (if any) result in properties of type `rtm:indicatedBy`. The purpose of stating that topics are of type `rtm:topic` seems to be the desire to use `rtm:topic` as an element type name in order to aid readability.

Associations are represented as blank nodes whose type corresponds to the association type. In addition, for each role in the association there is one statement whose property corresponds to the role type; its value is a node of type `rtm:member` that references the role player. Referencing is done through an `rtm:indicatedBy` property when the role player has a subject identifier and an `rtm:constitutedBy` property when the role player has a subject locator. (The text does not state what form the reference takes when the role player has neither.)

Having presented the methodology for translating XTM to RDF, Ogievetsky considers round-tripping from RDF to XTM and back to RDF. [**?**] is actually a continuation of earlier work for which only a set of slides ([**?**]) is available. In the earlier work RDF data was translated into XTM, again using XSLT stylesheets.

To demonstrate round-tripping [**?**] shows an example of a Dublin Core fragment in RDF being translated to XTM according to the methodology in [**?**], and then translated back to RDF according to the methodology in [**?**]. The source document contains a single RDF statement asserting that the resource ZARA.xml has the creator "Jane M. Folpe". This translates to a topic map consisting of six TAOs (five topics and one association), which in turn trans-

lates back to RDF as a set of no less than 26 RDF statements. "Obviously we accumulated a lot of semantic luggage during our roundtrip" is Ogievetsky's laconic comment.

The proposal appears to be fairly complete in that it covers more-or-less every aspect of XTM syntax (which requires extending the underlying PMTM4 model in order to cater for identifiers). The example of round-tripping shows clearly that this proposal in combination with the undocumented RDF2TM translation fails the test of reversibility since a single RDF statement ends up as 26 statements after the roundtrip.

The proposal requires seven statements to represent information content that would naturally be modelled using one statement in RDF and thus rates very low in terms of naturalness.

The Garshol Proposal

This proposal was originally presented in [**?**] as part of a comparative analysis of the RDF and Topic Maps models. The analysis was further developed (and extended to partially address OWL) in [**?**]. The approach has been implemented by the author in the Ontopia Knowledge Suite.

[**?**] starts by comparing RDF and Topic Maps through an examination of concepts that are fundamental to both paradigms: "symbols and things", "assertions", "identity", "reification", "qualification", and "types and subtypes". For each concept, Garshol shows how they are expressed in each paradigm and draws out the similarities and differences.

According to Garshol, RDF and Topic Maps are both "identity-based technologies"; that is, the key concept in both is symbols representing identifiable things about which assertions can be made. In Topic Maps, "things" are called "subjects"; in RDF they are called "resources" and, despite different definitions, they are essentially the same concept. Subjects are represented by topics; resources are represented by RDF nodes (or "nodes" for short). According to Garshol, the correspondence between "topic" and "node" is close but not exact; he does not explain why, but the reason is presumably that RDF nodes can be literals, which would be represented as strings rather than topics in Topic Maps.

Assertions express relationships between things and take the form of "topic characteristics" in Topic Maps and "statements" in RDF. A topic characteristic can be a name, an occurrence, or an association. An RDF statement can thus in theory be mapped to any one of these three kinds of construct. Special attention is paid to associations since these can be of any arity, whereas all

RDF statements are binary. A binary association maps fairly well to an RDF statement, but a non-binary association does not.

In addition, RDF statements have direction but associations do not. Topic Maps uses the notion of "roles" to express the nature of each subject's involvement in the relationship; in RDF this involvement is implicit in the subject-predicate-object structure of the statement.

For these reasons, the correspondence between topic characteristics and statements is considered to be close, but not exact.

The issue of identity is considered to be "quite a thorny problem for interoperability between topic maps and RDF" since, although Topic Maps and RDF both use URIs as identifiers, they do so in different ways. In RDF there is only one kind of identifier and a node can have at most one of them (blank nodes and literals have none). In Topic Maps, topics can have any number of identifiers and a distinction is made between "subject locators" (URIs which identify the subject directly, formerly called "subject addresses") and "subject identifiers" (URIs which identify the subject indirectly, via a subject indicator). Garshol refers the reader to a more in-depth treatment of the issue in [**?**].

Garshol's discussion of reification brings out certain differences between Topic Maps and RDF but does not reach any conclusion regarding the degree of correspondence between the two, although the point is made that reification is not a very commonly used feature. Qualification is seen as being related to reification, since the built-in Topic Maps feature "scope" is essentially a mechanism for making certain kinds of assertions about other assertions, but no proposal is made regarding how to express scope in RDF.

The concept of types and subtypes, on the other hand, is regarded as being identical in Topic Maps and RDF (except for the fact that the `subClassOf` property is part of RDF Schema rather than RDF itself).

Garshol summarizes his analysis by pointing to three fundamental differences between RDF and Topic Maps that "make it technically very difficult to merge" the two paradigms: identity, assertions, and reification (including qualification). The rest of his paper therefore focuses on ways to "move data between the two with as little effort as possible" (rather than on how to unify the two models).

The object mapping approach taken by [**?**], [**?**], [**?**], and [**?**] is briefly considered and then rejected as being both heavy-weight and rather awkward to work with. Any query or retrieval specified in end-user terms will have to explicitly take into account topic map model features, and information from topic maps will not interoperate cleanly with other RDF information.

Garshol's conclusion is that "although this [object mapping] approach is

easy to use, the results do not meet the criterion of clean integration with other RDF data."

As an alternative, Garshol proposes to use vocabulary-specific mappings underpinned by a generic mapping. Statements should in general be mapped to names, occurrences or associations since this provides the most "natural" results. However, it is not possible to know which of these is most appropriate for any given statement without an understanding of the semantics of the property in question hence the need for vocabulary-specific mappings. For example, the RDF statement:

```
<http://example.com/X>
  <http://example.com/Y>
    "foo" .
```

could be mapped in Topic Maps to either a name or an internal occurrence (since the object is a literal). Similarly, the statement:

```
<http://example.com/X>
  <http://example.com/W>
    <http://example.com/Z> .
```

could be mapped to either an association or an external occurrence (since the object is a resource). An optimal semantic translation cannot be performed without knowledge of the semantics of the properties Y and W.

The solution according to Garshol is to provide additional mapping information. This is done using an RDF vocabulary called RTM ([**?**]) which is used to annotate RDF documents (or their schemas) and thus guide the translation process. The RTM vocabulary is used for translating from RDF to Topic Maps and consists of the following RDF properties: maps-to, type, in-scope, subject-role, object-role.

The maps-to property can have the following values:

`rtm:basename` maps a statement to a base name

`rtm:occurrence` maps a statement to an occurrence

`rtm:association` maps a statement to association

`rtm:instance-of` maps a statement to a class-instance association

`rtm:subject-identifier` maps the values of a statement to a subject identifier

`rtm:subject-locator` maps the values of a statement to a subject identifier

`rtm:source-locator` maps the values of a statement to a source locator

Mappings that use `rtm:occurrence` or `rtm:association` will automatically use the statement's property to type the resulting Topic Maps construct, unless `rtm:type` is used to override this behaviour. The `rtm:in-scope` property can be used to specify scoping topics for base names, occurrences, and associations. Finally, the `rtm:subject-role` and `rtm:object-role` properties are used to specify the types of role played by the subject and object of an RDF statement when the statement maps to an association.

The vocabulary (and the implementations) go somewhat beyond what is covered in [**?**]. For example, it is recognized that properties may be mapped to various kinds of identifiers (source locators, subject identifiers, and subject locators) or to the privileged instance-of relationship, in addition to names, occurrences, and associations.

In addition, greater provision is made in the implementation for defaulting. Resource URIs are always mapped to subject identifiers and RDF statements can be imported as associations in the absence of role type information, in which case the predefined topics subject and object are used as role types.

Going from Topic Maps to RDF is shown to require additional information in order for optimal and/or predictable results to be achieved. The following problems are identified:

Choosing properties when mapping names Choosing the subject when mapping associations Garshol points out a number of issues that are not addressed in his analysis, including multiple identifiers, n-ary associations, reification and scoping, unary associations, variant names, and a number of (unspecified) "tricky edge cases"; for some of these he sketches possible solutions which have since been implemented:

Multiple URIs for the same topic can be handled using the RDF properties for equivalence found in OWL.

Associations with more than two roles can be turned into resources whose type is the association type, and each role can then be represented as a separate statement with the role type as the property and the association resource as the subject.

Reification and scoping can in general be represented by using RDF reification to represent the statement that would connect the topic characteristic with the topic. A special property will have to be defined for representing scope.

As for the reification this is done by simply merging the resource representing the topic characteristic assignment with that representing the reifying topic.

Binary non-symmetric associations can be handled by having the mapping contain one association from the association type to the preferred subject role.

Selection of name properties can be done by having the mapping contain an association from the topic type to a topic representing the preferred RDF name property.

A second vocabulary (called TMR, [**?**]), consisting of six published subjects, addresses many of these issues. Name mapping is handled by tmr:name-property, tmr:type, and tmr:property, and the problem of mapping associations is solved using tmr:preferred-role, tmr:association-type, and tmr:role-type.

As with the RDF2TM translation, the implementations provide some level of defaulting. Both subject identifiers and subject locators are automatically mapped to resource URIs. In addition, associations can be exported to RDF in the absence of mapping information about roles; in this case the choice of subject and object for the resulting statement is arbitrary.

As currently specified the Garshol proposal provides an almost complete solution and the author himself identifies most of the respects in which it is incomplete. Those which are not mentioned include containers, collections, XML literals and typed literals. A high degree of reversibility and round-tripping is achievable, provided appropriate reverse mappings are generated during the translation. An issue exists with subject locators that end up as subject identifiers when round-tripping from Topic Maps to RDF and back to Topic Maps.

The proposal scores well in terms of naturalness. Even when using default mappings the results are quite natural. The TM2RDF test case results in an RDF document containing 13 statements. The RDF2TM test case results in a topic map containing nineteen topics, three associations, and three occurrences.

The Unibo Proposal

Ciancarini et al cite the work of Moore, Lacher and Decker, and Ogievetsky, all of which, they claim, suffers from a common drawback, namely the "rather awkward appearance of the documents coming out of the conversion." The authors clearly prefer Garshol's approach, which produces much more "readable" results and which is similar to their own. The main difference is that Garshol does not utilize the "standard RDF and RDFS predicates" and thus always requires a mapping to be specified.

Like earlier authors, Ciancarini et al recognize that there are two fundamen-

tal approaches to tackling the problem of translation, corresponding to what this survey calls object mapping and semantic mapping. The first of these is seen to be problematic in that "the converted document is necessarily very different from the one that would have been written directly in the destination language, and hardly readable." The problem with the second one is that it is "not always possible" to identify semantic equivalences, and that doing so often requires a case-by-case approach and thus has no general usefulness.

The authors therefore consider a hybrid approach to be the optimal solution and their implementation in the Meta Converter combines a generic mapping, which tries to stay as close as possible to the original semantics, with the ability to define specific mappings using an XML vocabulary.

Identity

Like Garshol, Ciancarini et al assume a basic equivalence between topic and resource (although they are less clear on the distinction between resources and RDF nodes), but they differ in how identity is expressed. The default behaviour in the Unibo proposal is to equate subject locators with resource URIs and to represent subject identifiers using the RDFS property isDefinedBy. Examples given in [**?**] (e.g., 3.8 and 4.2) show how this leads to resources that clearly represent non-addressable subjects, such as "Mario Rossi" and "Format", being translated to addressable subjects (using ¡resourceRef¿ for subjectIdentity).

Topics that have no subject locator are translated to blank nodes whose ID is generated from the topic's base name. When going the other way, the ID of a blank node becomes a topic name, which is clearly unnatural (since the ID of a blank node and a topic name have different semantics).

Names

The Unibo proposal is alone in assuming a fundamental equivalence of semantics between base names and the rdfs:label property: Names that have no variants are thus easy to handle. Variant names are seen to represent a greater challenge which is solved through the use of four RDF predicates: baseName, variant, parameter, and variantName. A base name that has a variant is represented through a blank node with rdfs:label and tm2rdf:variant properties: the former is a literal that corresponds to the value of the topic name (i.e., the ¡baseNameString¿ in XTM syntax); the value of the latter property is another blank node that has variant and parameter properties. Thus a topic with a base name and sort name:

`[mario_rossi = "Mario Rossi";"rossi mario"] results in the following statements:`

```
_:mario_rossi
  tm2rdf:baseName    _:bn1_mario_rossi .

_:bn1_mario_rossi
  rdfs:label         "Mario Rossi" ;
  tm2rdf:variant     _:v11_mario_rossi .

_:v11_mario_rossi
  tm2rdf:variantName "rossi mario" ;
  tm2rdf:parameter   _:param1 .

_:param1
  rdfs:isDefinedBy   <http://www.topicmaps.org/xtm/1.0/core.xtm#sort> .
Associations: TM2RDF
```

Predictably, representing associations in RDF is regarded as difficult because of what the authors term RDF's "more primitive" (i.e., low level) nature compared to Topic Maps. A generic translation is possible "at the level of the model," but it is "complex and artificial" and comes at the price of "abusing the RDF way of expressing relationships." The basic approach is similar to Ogievetsky's in that the roles (or "members") are contained in an RDF bag of blank nodes. However, whereas in Ogievetsky the bag is the association, the Unibo proposal uses an additional resource to represent the association; this resource has a tm2rdf:association property, the object of which is the bag of members. All in all, nine RDF statements are required to represent a single binary association.

The tm2rdf:association property is characterized as a "supporting predicate" whose purpose is to "add a little legibility" to the resulting document. A variation on this is also suggested in which the bag of members and the association become a single node: This is effectively the same solution as Ogievetsky's.

[**?**] also describes two alternative approaches in which n-ary associations are decomposed into a number of binary relations. Both of these require six RDF statements in order to represent a single ternary association. Given the following association:

X(A : rA , B : rB , C : rC) (i.e. an association of type X between topics A, B, and C playing the roles rA, rB, and rC respectively), the first of these alternative approaches results in the following six statements:

A X B . A X C . B X A . B X C . C X A . C X B . Role types are lost. In addition, the fact that each pair of role players is related through the same

predicate twice (both as subject and object and as object and subject) means that only symmetrical relationships would be represented correctly. Finally, the semantic of A, B, and C all being involved in the same relationship is also lost; this may or may not involve real loss of information depending on the nature of the relationship.

The second alternative approach involves predicates that correspond to role types and results in the following statements:

B rA A . C rA A . A rB B . C rB B . A rC C . B rC C . While role types are now preserved, the association type is lost (although it could in theory be preserved through additional statements relating it to rA, rB, and rC). In addition, it seems doubtful that the original semantics are correctly preserved. For example: Can it be assumed to be the case that the relationship between role players B and A (rA) is the same as that between C and A? Finally, the point made above about losing the semantic of the involvement of A, B, and C in the same relationship also pertains here.

Having considered these alternatives, the Unibo proposal comes down in favour of the approach that uses the tm2rdf:association property, at least in the absence of more specific mapping information.

Associations: RDF2TM

When translating in the opposite direction, from RDF to Topic Maps, the generic solution proposed by Unibo is to translate RDF statements to associations. The example given in [**?**] results in a typed binary association with untyped roles and does not take into consideration the case in which the object of the RDF statement is a literal. However, it is conceded that "it might be preferable, in certain contexts, to apply other types of conversion" and this leads into a discussion of "attributes" and the role of schema information.

The Unibo proposal recognizes that certain RDF statements are more appropriately mapped to either internal or external occurrences, with the occurrence type corresponding to the property of the statement, but knowing when to do this requires some kind of schema information. This is essentially the same as Garshol's approach, except for the fact that Unibo uses an XML vocabulary rather than an RDF vocabulary to specify the mapping information.

Scope

In this context a proposal is put forward for representing scoped occurrences in RDF: An rdfs:seeAlso property has a blank node as its object; the blank node has an rdfs:isDefinedBy property (whose object is the URI of an external occurrence) and one or more tm2rdf:scope properties. This results in a construct whose "shape" is very different from that of an unscoped occurrence. In addition, given that the range of the rdfs:isDefinedBy property is rdf:Resource,

it is unclear how this approach would work with internal occurrences.

A "not very elegant" way to represent scoped names is suggested that involves defining a property, whose rdf:type is tm2rdf:baseName, that corresponds either directly or indirectly (it is not clear which) to each scoping topic. In addition to being inelegant, this would not work with scopes comprised of multiple scoping topics. The alternative is the same as that proposed by Garshol: i.e., to qualify reified statements. To do this, Unibo defines the tm2rdf:scope property.

For scoped associations, reification in the RDF sense is not necessary since associations are already represented as resources (at least in the default mapping). Thus, all that is required is to assign one or more tm2rdf:scope properties to that resource. The downside to this is that scoping is now handled in three different ways (for generically mapped associations, for occurrences, and for names and specifically mapped associations, respectively).

Reification, typing, and subtyping

Neither reification, typing, or subtyping are regarded by Unibo as posing problems since both RDF and Topic Maps support all three concepts in essentially the same way: instanceOf equates to rdf:type; the supertype-subtype relationship (represented in Topic Maps using an association with a predefined type) equates to rdfs:subClassOf, and reification is essentially the same in Topic Maps and RDF.

Specific mappings

The description above has focused on the Unibo proposal's approach to generic translations. However, it is recognized that a generic approach will not always produce optimal results and so a method is provided for "guiding" the translation process. This consists of a simple XML vocabulary that allows the user to specify how to translate a (binary) association to a single RDF statement (and vice versa). As in the Garshol proposal, this involves specifying correspondences between association role types and the statement's subject and object. In addition, a user can specify which RDF properties should be mapped to occurrences rather than to associations. The following extract shows how mappings for the TM2RDF test case would be specified:

```
<?xml version="1.0"?> <xtm2rdf>
  <property_associations>
    <li id="composed-by">
      <domain_role id="work"/>
      <range_role id="composer"/>
    </li>
```

```
  </property_associations>
  <property_occurrences>
    <li id="premiere-date"/>
    <li id="synopsis"/>
  </property_occurrences>
</xtm2rdf>
```

These mappings would cause the composed-by association to be represented as a single statement in RDF, with Tosca ("work" = domain) as the subject and Puccini ("composer" = range) as object. In addition, the mapping contains information that would cause properties of type premiere-date and synopsis to be mapped to occurrences when going from RDF to Topic Maps. (Although not stated explicitly, this information is presumably not required when going the other way.)

The Unibo proposal is fairly complete but some features, e.g., language tags and data typing in RDF, and reification of roles and topic maps, are not covered explicitly. The proposal permits some degree of reversibility, but the result of a roundtrip may not always be the same as the starting point. For example, using the generic mappings, most RDF statements would be translated to typed associations with untyped roles, each of which would result in several statements when translated back to RDF.

The approach produces somewhat natural results in both directions provided mapping information is supplied. Generic translations are far less satisfactory, with a single binary association resulting in nine RDF statements.

3.5 Extending MIDST to Semantic Annotations

In Chapter 2 we have illustrated the MIDST approach on translation between heterogeneous models, exploiting a metamodel technique, developed by our group. In this section we will show how the MIDST framework can be suitably extended to solve interoperability issues between semantic Web languages in different domains, such as the semantic annotations formalisms RDF and Topic Maps. In the MIDST approach all the models, involved in the translation process, are described by means of the higher level, generic model (the supermodel) that builds on metaconstructs. A metaconstruct has properties allowing the description of models constructs and references to other metaconstructs to define the relationships between them. The expressivity of our approach can be flexibly extended by adding new metaconstructs to the supermodel or just adding properties or references to the existing ones.

The supermodel has been originally defined with respect to a variety of models, for which translations can be applied by our approach (E-R, Relational, Object-Oriented, Object-Relational, XSD). The supermodel, as it is, isn't enough expressive to describe Semantic Web formalisms, where there are some specificities that distinguish them from the data models we have involved till now in our approach. One of the most interesting characteristic to take into account is the generality of some of the constructs that are used in Semantic Web models. Looking specifically at the two formalisms we consider, RDF has the concept of resource, that can be a web resource, a predicate, a specific person, the concept of person and all the things that can be described. A similar idea is at the basis of Topic Maps where (quite) everything is a topic, like the described subjects, the types, the association roles etc. Therefore, RDF and Topic Maps are underpinned on the knowledge element, that is described by resources and topics respectively, and everything is considered knowledge. This causes the collapse of some of the *instance level* and *conceptual level* information. To introduce the ability to express schema information other models have been developed, RDF(S) and OWL extending RDF, TMCL extending Topic Maps. Differently, the data models that we have considered till now, have a well marked distinction between the different layers: schema, instances. For example, in the database context we have the concept of *person* with attributes at the schema level and the specific values for the attributes at the instance level. To clarify which is the impact in our approach, let's see how MIDST can be used to represent the Relational table *Person(Name, Surname)* with the instances *George, Russel* and *John, Smith*.

Table 3.2: Relational table Person

Name	Surname
George	Russel
John	Smith

Referring to the supermodel representation of the Relational model, the `Abstract` metaconstruct is used to represent the Relational model construct `Table` that can be used to define the schema element `Person`. By means of the metaconstruct `Lexical` it is possible to describe the Relational model construct `Attribute` (of the table `Person`). The table `Person` is used to define the schema elements `Name`, `Surname` and at the instance layer there are the actual attributes values. In the RDF/Topic Maps context, you do not start from a schema level, that act as a constrain on the instance level but the

starting point are the resources/subjects that needs to be described with an extreme freedom. The knowledge expert can choose how to use the model's constructs to describe resources. So for example in RDF you will start from a blank nodes representing the actual persons and linking them with a property `rdf:type` that has object the node `Person`, then adding the properties `Name` and `Surname`. The possible RDF representation, using the FOAF vocabulary, is shown in the next chunk of code and the correspondent graph is sketched in Figure 3.6.

```
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:foaf="http://xmlns.com/foaf/0.1/">
<foaf:Person>
  <foaf:name>John</foaf:name>
  <foaf:surname>Smith</foaf:surname>
</foaf:Person>
<foaf:Person>
  <foaf:name>George</foaf:name>
  <foaf:surname>Russel</foaf:surname>
</foaf:Person>
</rdf:RDF>
```

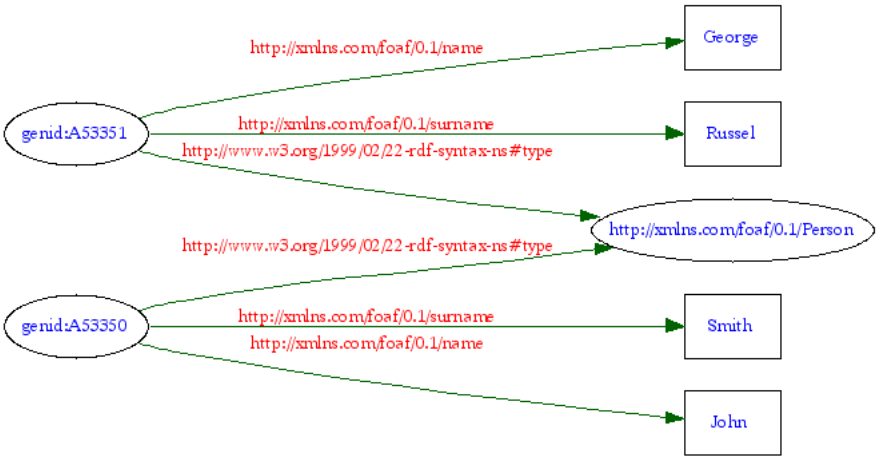


Figure 3.6: RDF example for the relation Person

Nevertheless, it is also valid to express only the information about name and surname, without specifying that the elements we are describing are of type person. The RDF code, that can be produced with respect to this modification, is reported in the following and the correspondent graph is sketched in Figure 3.7.

```
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:foaf="http://xmlns.com/foaf/0.1/">
<rdf:Description>
  <foaf:name>John</foaf:name>
  <foaf:surname>Smith</foaf:surname>
</rdf:Description>
<rdf:Description>
  <foaf:name>George</foaf:name>
  <foaf:surname>Russel</foaf:surname>
</rdf:Description>
</rdf:RDF>
```

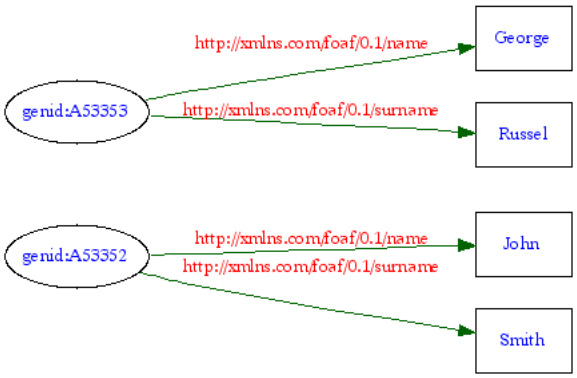


Figure 3.7: RDF example representing only name and surname

The meaning of the above example is that there is someone (or something) with name John and surname Smith, and someone (or something) with name George and surname Russel. In this case, nowhere is specified that the Name and the Surname are attributes of a Person but they are directly related to the instances. In our approach we cannot have attributes (lexicals) without a

related concept (abstract). Since we operate at different levels of abstraction, we need to extract a sort of structure from the RDF and Topic Maps graphs that is suitable with our way of representing models, schemas and instances.

To enable the supermodel representing RDF and Topic Maps, reflecting the above mentioned peculiarities, we have enriched some of the existing metaconstructs and new metaconstructs are introduced.

Figure X illustrate the extended supermodel, red elements corresponds to the added features.

In the following there is a brief description of the extended supermodel's metaconstructs, details about the specific RDF and Topic Maps representation will be discussed later.

Abstract

The meaning of `Abstract` is extended to represent the symbols that are used to describe the subjects of interest in RDF and Topic Maps. With the term `Abstract` we generically describe the nodes of an RDF graph as well as the topics of a Topic Map. To specify the node (resource or topic) properties, as if it would be a resource with an URI or a Literal, we use the `Type` property of `Abstract`. This allows us to define a generic transformation between the structure of the graphs.

BinaryAggregationOfAbstracts

With `BinaryAggregationOfAbstracts` we can describe the Topic Maps `Occurrence` and `Name` relationships extending it with an optional reference to an abstract that represents the scope (that we generically call *Qualification*).

AbstractAttribute

This metaconstruct is used to describe the properties of RDF. Another metaconstruct that could candidate to describe RDF properties could be the **BinaryAggregationOfAbstracts** but this is used to represent the binary relationship that are bidirectional. For this purpose, this metaconstruct is endowed with the ability of expressing the functionality and optionality in both directions. The `AbstractAttribute`, instead, is used to assert that an `Abstract` is linked to another `Abstract` via a property, that is, the name of the `AbstractAttribute`. This is exactly correspond to RDF properties.

Lexical

The `Lexical` metaconstructs are used to express the different kinds of identification that Topic Maps allows, namely `SubjectIndicator` and `SubjectAddress`.

AggregationOfAbstracts
The `AggregationOfAbstracts` has been extended with two optional references to abstracts that represent the type and the qualification of the aggregation, this is used for the n-ary `Association` of Topic Maps. The type and qualification are expressed as abstracts since they are topics.

ComponentOfAggregationOfAbstract
Each `ComponentOfAggregationOfAbstract` of the extended supermodel has a reference to an abstract that is the role of the participation, a topic itself.

Type
The introduction of the new metaconstruct `Type`, which expresses the typing relation between abstracts, allows us to model the RDF construct `rdf:type` and the equivalent for Topic Maps that is `instanceOf`.

Set
This is a new construct we have introduced in the extension, is used to map the collections and containers of the RDF model. The attribute `Type` allows the distinction between the different kind of sets.

ComponentOfSet
This metaconstruct references the `Abstract` with the `Set` of which is a component.

Figures 3.8 and 3.9 gives an idea of which constructs are used to represent RDF and Topic Maps, respectively. We don't aim at covering completely all the constructs of the two considered models, since we demonstrate how the supermodel is extensible. We here want to demonstrate that our approach is suitable to work with so different kind of models.

We have provided the supermodel with these new capabilities through which we can now illustrate how to perform a translation between Topic Maps and RDF.

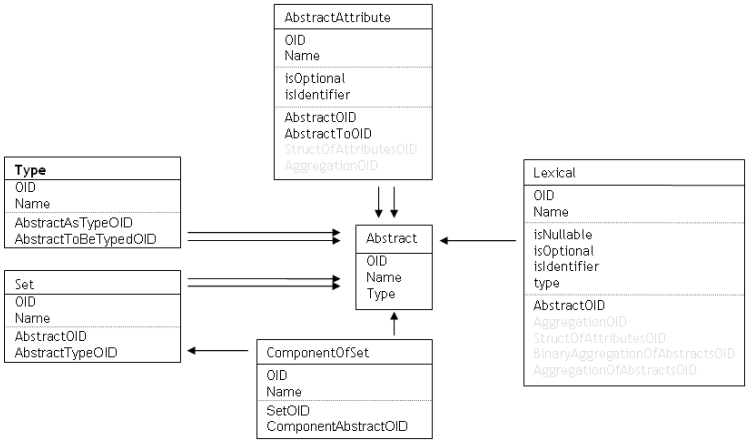


Figure 3.8: The supermodel's constructs used to represent RDF

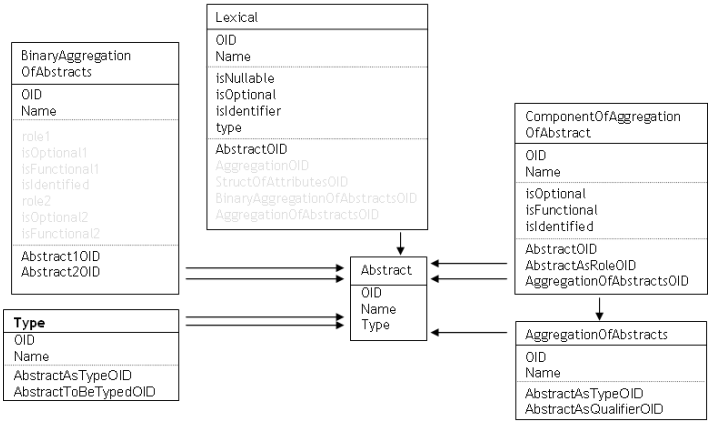


Figure 3.9: The supermodel's constructs used to represent Topic Maps

3.6 Translation between RDF and Topic Maps

As illustrated in Section 3.4, there are many ways to implement interoperability between Topic Maps and RDF: creating virtual views that make RDF appear to be Topic Maps (or vice versa) when viewed through a particular interface

(object mapping) or doing a conversion of data from one formalism to the other by means of a somehow expressed mapping (semantic mapping).

Devising a generic mapping can sound as the ideal way to perform translations, avoiding to force a model to behave as the other, implementing translations in a more natural way. But when we create the mappings between RDF and Topic Maps, it arises the problem of a lack of information because of the different expressivity of the two models. Let's say that we want to translate the RDF statement `<S,P,O>` in Topic Maps. It is an assertion that can be translated in an *Association*, an *Occurrence* or a *Name* and it cannot be known a priori which is the right choice. We need to evaluate the meaning of the property P to select which is the right target construct. The result is a vocabulary-specific mapping that nevertheless allows us to reach satisfiable results in terms of keeping the expressiveness of the model and its meaning during the translation process.

As we demonstrate with the example below, the use of Datalog to write translation rules allows us to naturally implement the vocabulary-specific mappings by means of conditional expressions that select the correct target construct. Moreover rules are written at the metalevel between the representations of source and target model in terms of metaconstructs, keeping the independence from a given model and allowing MIDST to involve new models reusing the previously made work.

The first step when translating with MIDST is to describe the source and target model's constructs in terms of the supermodel metaconstructs, then to define the source schema that can be imported into the supermodel by a copy operation. Following the correspondences presented in section 4 we can represent both RDF and Topic Maps using the supermodel and write the translation rules at the metalevel.

Let us consider the simple RDF document that follows that illustrate the knowledge between two persons one of which with a homepage, using the foaf vocabulary.

```
<rdf:RDF
 xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
 xmlns:foaf="http://xmlns.com/foaf/0.1/">
    <foaf:Person>
      <foaf:name>John Smith</foaf:name>
      <foaf:homepage rdf:resource="http://www.john.sm"/>
      <foaf:knows>
        <foaf:Person>
          <foaf:name>Fred Red</foaf:name>
        </foaf:Person>
      </foaf:knows>
    </foaf:Person>
</rdf:RDF>
```

The related graph in sketched in figure 3.10:

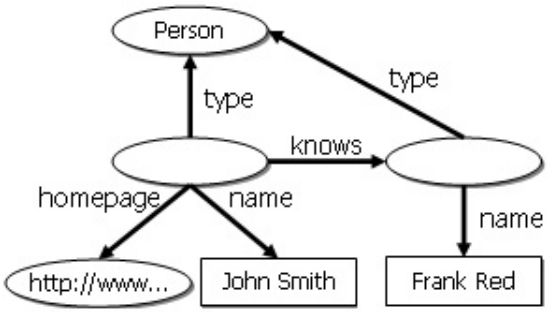


Figure 3.10: RDF Example

The supermodel representation of the above document, stored in our relational dictionary, is illustrated in Fig. 3.11.

It can be noticed that while in the `AbstractAttribute` table there is the specification of the actual names of the properties, the `Abstract` table stores generic RDF nodes and the `Type` columns specificate the kind of node. The values of the literals and URIs are not relevant during the translation and are represented at the instance level.

When translating the RDF statements into Topic Maps, the Datalog rules generate the constructs

Abstract		
OID	Name	Type
1	RDF_Node	URI
2	RDF_Node	URI
3	RDF_Node	Literal
4	RDF_Node	Literal
5	RDF_Node	Blank
6	RDF_Node	Blank

AbstractAttribute			
OID	Name	AbstractOID	AbstractToOID
1	rdf:type	2	1
2	foaf:knows	2	3
3	foaf:homepage	2	4
4	foaf:name	2	5
5	rdf:type	3	1
6	foaf:name	3	6

Figure 3.11: MIDST Relational dictionary storing an RDF document

- `BinaryAggregationOfAbstracts` for the property `foaf:name`, that will subsequently be mapped to `Name`
- `BinaryAggregationOfAbstracts` for the property `foaf:homepage`, that corresponds to `Occurrence`
- `AggregationOfAbstracts` for the property `foaf:knows`

For the last case, we don't have the information of the association roles in RDF while Topic Maps misses the direction of the statement. We choose to assign the roles values of `rdf:subject` and `rdf:object` to the association participants, in order to keep the semantic of the direction of the RDF arc. As result of the translation we obtain the following Topic Maps, the syntax we use is XTM:

```
<topic id="id18">
 <instanceOf>
  <subjectIndicatorRef xlink:href="foaf:Person"/>
 </instanceOf>
 <baseName>
  <baseNameString>John Smith</baseNameString>
 </baseName>
 <occurrence>
  <instanceOf>
   <subjectIndicatorRef xlink:href="foaf:homepage"/>
  </instanceOf>
  <resourceRef xlink:href="http://www.john.sm"/>
 </occurrence>
</topic> <topic id="id32">
```

```
 <instanceOf>
  <subjectIndicatorRef xlink:href="foaf:Person"/>
 </instanceOf>
 <baseName>
  <baseNameString>Frank Red</baseNameString>
 </baseName>
</topic>
<association>
 <instanceOf>
  <subjectIndicatorRef xlink:href="foaf:Knows"/>
 </instanceOf>
 <member>
  <roleSpec><subjectIndicatorRef
                    xlink:href="rdf:subject"/></roleSpec>
  <topicRef xlink:href="#id18"/>
 </member>
 <member>
  <roleSpec><subjectIndicatorRef
                    xlink:href="rdf:object"/></roleSpec>
  <topicRef xlink:href="#id32"/>
 </member>
</association>
```

Some of the rules, that we employed during the translation are reported in Appendix B.

On the other side, considering the translation from Topic Maps to RDF, specifically the case of transforming an association, we have to transform n-ary relationships to binary statements. In Fig. 3.12 there is a topic map that represents the employment association between the company HiTech, with the role of employer and two employees.

To translate this map to RDF, we define a Datalog rule that creates an `Abstract` for each topic, with lexicals that represent the identification and the names. Other simple rules are devoted to the creation of an `Abstract` without lexicals (blank node) for each participant, that allows us to define `BinaryAggregationOfAbstracts` that link the participant topic with the role topic. We then create an `Abstract` for the association and an `Abstract` for the type that are linked by a `Type` metaconstruct. Finally members are linked to the association by `BinaryAggregationOfAbstracts`. Exploiting the correspondence between the supermodel and RDF we can eventually generate the

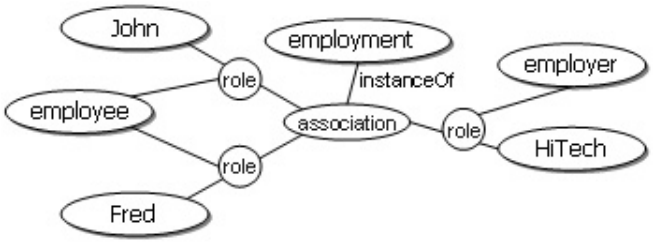


Figure 3.12: Topic Maps example

graph as illustrated in Fig. 3.13.

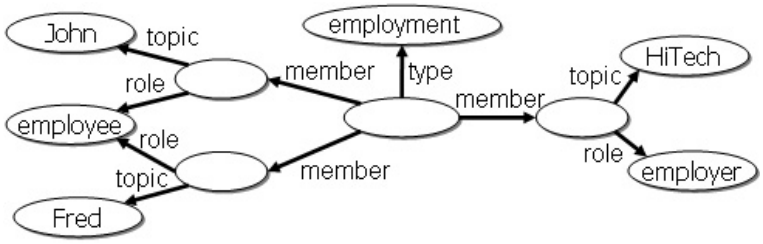


Figure 3.13: Resulting RDF graph

The results of the experiments produced in this field have been published in [**?**].

3.7 Conclusions

We have illustrated how Semantic Web formalisms can be described by our supermodel and how our framework can be used to define translations with a high-level, model independent approach. Data models and schemas are currently defined using the framework interface, we are developing a plug-in that allows to automatically import external documents of the source model inside the tool and to export the result of the translation in the target model. Moreover, we are developing a visual interface that allows users to query documents exploiting our logical organization.

Chapter 4

A Scalable and Extensible Framework for the Management of RDF data

The Semantic Web is gaining interest to fulfill the need of sharing and reusing information. In this context, RDF has been conceived to provide an easy way to represent any kind of data and metadata, according to a graph-based, lightweight model and a straight XML serialization. Although RDF has the advantage of being general and simple, it cannot be used as a storage model as it is, since it can be easily shown that even simple management operations involve serious performance limitations.

Starting from an analysis of the current state of the art for RDF data management, we present in this thesis a novel approach for storing, managing and processing RDF data in an effective and efficient way. The approach is based on an organization that is particularly suited for RDF constructs, but it can be easily extended to other models that relies on RDF, like RDFS and/or OWL. We refer to real world scenarios where large RDF data repositories need to be navigated and processed, even if the schema is not known in advance. We consider complex data management operations, which go beyond simple selections or projections, and involve the navigation of huge portions of data sources.

A tool has been developed to test the feasibility of the approach over large RDF repositories (datasets of millions of triples). We compare the performance of the proposed framework with prior solutions and present experimental results

supporting its effectiveness and efficiency.

4.1 Introduction

From the origins of the World Wide Web the activity of publishing information has been so intense that a huge amount of data is currently available over the internet. The technologies that have become standards for the creation of Web content are oriented to human users. Moreover, the absence of a unique point of reference for the semantic of data do not allows the use of software agents that understand the meaning of data, performing advanced searches, inferencing, wrapping, etc. The interpretation of the meaning of data is competence of the reader, with the obvious consequence of misunderstandings. This initial approach reveals a lack for the share and reuse of information, needs that continuously grow up with the increasing of available data over the Web.

In 2001 Tim Berners Lee [**?**] proposed the concept of Semantic Web, a world where information is processable by machines other than humans, opening new interesting perspectives. The principle to let a rough data be unambiguously interpreted by a computer, is to describe it by means of metadata that associate a well defined meaning. For this purpose the W3C developed the Resource Description Framework (RDF) [**?**] that is commonly used as the data model for the Semantic Web. It is a family of specifications used to express statements on Web resources, uniquely identified by an URI (Uniform Resource Identifier). The statement is in the form of a triple $\langle Subject, Predicate, Object \rangle$ expressing that a resource (Subject) is related to another resource or to a value (Object) through a property. An RDF document can then be seen as a directed labeled graph where nodes are resources or literals and arcs represent predicates, also called properties in RDF terminology. Nodes without content (blank nodes) are used in order to express incomplete information or queries or resources with a complex structure (specified via other triples). Many serializations of the abstract model of RDF exist, to allow the exchange of metadata across applications, the most diffused are RDF/XML and Notation 3. To provide a datatyping model for RDF, the W3C defined RDF Schema that allow the knowledge designer to define classes with subclasses, properties with range and domain, containers and other elements to define a slight more advanced knowledge structure.

Due to its simplicity RDF is easily manageable by applications and increasingly used over the Web. However the rise of Semantic Web technologies requires a growing number of meta-information to be managed, transformed

and queried. The maintenance and querying of RDF documents represent crucial activities to profit from available semantic information. The concept of maintenance is related to the insertion or deletion of a node in an RDF document. Insertion operations add new knowledge to the RDF document, through inference or explicit insertions. Deletion operations represent the elimination of information from the RDF document, adding some complex problems because of potentially dangerous operations due to existing dependencies between nodes. In this thesis we will consider maintenance related to scalability issues by applying our approach to a growing RDF document. Querying an RDF document is also an active field of research, as shown by the large number of languages developed to this aim (see [**?**] for a recent survey). To perform queries, RDF data can be stored in different kinds of repository. As we will detail in the following, proposals in this sense mainly concentrate on a triple organization of data with some tuning operations to improve query execution. Managing RDF triples in one table, however, compromise scalability and performances because of the high number of self-joins needed. To avoid this problem, some approaches present a different data organization. For example 3Stores [**?**] separates literals from resource values while in [**?**] different RDF storage schemes are considered (i.e. vertical partitioning, property tables, column oriented DBMS).

In this thesis we propose a storage system that is specifically suited to work with huge RDF documents as published in [**?**]. We address the issues of storing and querying RDF exploiting a metamodel technique. Differently from the approaches in literature, that provide implementation solutions, we follow a process that starts with the definition of a meta-representation of the RDF datamodel at a conceptual level. In this phase we find out the structural aspects of interest of RDF. The next step is a logical translation of the conceptual model for a relational database. Eventually, at the physical level, we choose optimization techniques based on indexing and partitioning. We here concentrate on pure RDF (with no schema information) but the representation technique that we use is extensible: the model can be enriched to represent structural aspects of more complex models like RDF(S) and OWL.

The approach we propose here can be used for an agile management of RDF documents. More precisely, the most relevant contributions of our work are:

(i) The creation of a high level description model to represent the information stored in RDF document, pointing out the implicit semantics of elements through constructs.

(ii) A logical organization of data, based on the constructs structure of the model, referring to a relational database.

(iii) Optimizations based on indexing and partitioning of relational implementation to allow high performance querying and mantaining.

We compare our experimental results with two other approaches: the standard RDF triple [**?**] and Vertical Partitioning [**?**]. The aspects that we have measured are the scalability of the approaches and query performance over a set of representative queries.

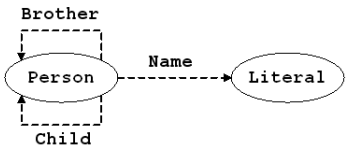
The chapter is structured as follows. In Section 2 we present a motivating example. In Section 3 we discuss some related works. in Section 4 we illustrate the details of our design process to manage RDF data. In Section 5 we show experimental results of performance and scalability and in Section 6 we sketch concluding remarks and future works.

4.2 Running Example

Commonly, an RDF document is organized in a set of triples that can be stored in one single relational table using a three-column schema. Let us consider the Figure 4.1 showing RDF classes (left side) and corresponding triple instances (right side) of an RDF document. We consider both properties (relations between a resource and a literal) and predicates (relations between two resources). More in detail, each class *Person* has a property, *Name*, and two predicates, *Child* and *Brother*, representing family relationships between persons. There are four instances of the class *Person* (with *URI1*, *URI2*, *URI3* and *URI4* as URIs[1], respectively), each one with a corresponding instance of the property *Name* (with values *Priam*, *Hector*, *Astyanax* and *Paris*, respectively) and linked by two instances of predicate *Child* and one instance of predicate *Brother* representing that *Hector* and *Paris* are sons of *Priam* and brothers, and *Astyanax* is son of *Hector*.

The RDF instance can be stored in a relational table `people`, as shown at the right side of Figure 4.1. We can perform various operations on the RDF data regarding both queries and update/maintainance. For example, we can formulate the SQL query to find all of the persons that have a child whose name is '*Paris*' as follows:

[1]RDF exploits Universal Resource Identifiers (URIs), which appear as URLs that often use sequence of numbers as identifier. In this thesis, our examples will use more intuitive names.



people

Subject	Property/ Predicate	Object
URI1	rdf:type	Person
URI2	rdf:type	Person
URI3	rdf:type	Person
URI4	rdf:type	Person
URI1	Name	Priam
URI2	Name	Hector
URI3	Name	Astyanax
URI4	Name	Paris
URI1	Child	URI2
URI1	Child	URI4
URI2	Child	URI3
URI2	Brother	URI4

Figure 4.1: RDF classes and triple instances

```
SELECT p4.obj FROM people AS p1, people AS p2, people AS p3, people
AS p4 WHERE p1.prop="Child" AND p1.obj=p2.subj AND
p2.prop="rdf:type"
      AND p2.obj='Person' AND p1.obj=p3.subj AND p3.prop="Name"
      AND p3.obj="Paris" AND p4.subj=p1.subj AND p4.prop='Name'
```

This query performs many self-joins over the same table. Since the table `people` could contain a relevant amount of statements, the entire process potentially presents a high execution complexity. Indeed, the execution time increases as the number of triples scales, because each filtering or join condition will require a scan or index lookup.

Let us consider a more complex query. We want to find the family relation between *Astyanax* and *Hector* as follows:

```
SELECT t1.prop FROM people AS t1, people AS t2, people AS t3, people
AS t4,
     people AS t5
WHERE t1.subj=t2.subj AND t2.prop='Name' AND t1.subj=t3.subj
      AND t3.obj='Person' AND t1.obj=t4.subj AND t4.prop='Name'
      AND t1.obj=t5.subj AND t5.obj='Person' AND
      ((t2.obj='Hector' AND t4.obj='Astyanax')
      OR (t4.obj='Hector' AND t2.obj='Astyanax'))
```

Also in this case the number of self-joins increases as the number of scan or index lookups.

Real world executions does not only involve a lot of joins or filters, which make critical the selectivity and optimization of query (i.e. limiting the benefit of using indexes), but also maintaining operations as insertion, deletion and update of triples. The maintenance of large RDF datasets represents a complex problem because of potentially dangerous operations due to existing dependencies between nodes: it is important to identify, for instance, which other nodes must be deleted as the effect of a single deletion.

4.3 Related Works

Managing RDF information represents a important and wide area of research and a number of methodologies and techniques, along with many tools have been developed [**?**]. In this section we discuss the current researches concerning the management of RDF documents.

We distinguish among three main direction in RDF management based studies, namely: i) *storing*, that can be a complex activity, due to the fact that some storage systems requiring fixed schemas may be unable to handle general data such as that from RDF, where the terms are not known in advance; ii) *querying*, that might be seen as more complex than "conventional" querying, because the meaning conveyed by RDF data has to be properly "understood" and processed; iii) *maintaining*, that is the set of update/deletion operations on RDF documents.

In this thesis we mainly focus on storing, because through our representation it is possible to perform querying and maintaining of RDF documents via SQL statements.

RDF storing approaches can be divided into two main areas: the first one is based on native store system development while the second one focuses on the use of relational (or object-relational) databases to store RDF data. Comparing the two approaches, native store systems (such as AllegroGraph[2] or OWLLIM [**?**]) are more efficient in terms of load and update time. On the contrary, DBMS-based approach are more efficient in querying due to the availability of many query management features. However, native storing approaches have the drawback of having to redefine important database features such as: query optimization, transaction processing, etc.

To efficiently query RDF data is important to organize these information in a convenient way. Some interesting studies were focuses on the physical organization of RDF Data. Storing RDF in relational (and/or object-relational)

[2]AllegroGraph RDF-Store, available at http://www.franz.com/products/allegrograph/

database management systems has been the main topic of much research. Indeed it seems that a relational model is a natural fit for RDF. The RDF model is a set of statements about Web resources, identified by URIs. The statements have the form $< subject, predicate, object >$, then an RDF model can be easily implemented as a relational table with three columns. Many RDF storage systems have used this kind of approach such as the ICSFORTH RDFSuite [**?**], the Sesame schema-based repository and querying system [**?**], the KAON[3] Project, the TAP[4] suite, Jena [**?**], Oracle [**?**], and many others. Most of the aforementioned systems use storing techniques that do not involve entire strings in the triples table; instead they store shortened versions. Oracle and Sesame map string URIs to integer identifiers so the data is normalized into two tables, one triples table using identifiers for each value, and one mapping table that maps the identifiers to their corresponding strings. 3store [**?**] uses a similar approach, the only difference is on the fact that the identifiers are created by applying a hash function to each string.

Nevertheless, these approaches present various limitations. For instance, considering the NULL values (possibly many) management in case of blank nodes or the awkward expression of multi-valued attributes in a flatted representation. Therefore the typical triple store approach cannot leverage this higher-level knowledge. In order to overcome some of the aforementioned hindrances we propose an approach that can take into account also the blank nodes as described in the rest of the chapter.

For scalability and high-performance, we believe that the triple-store approach must be augmented by other storage strategies that can be efficient and sufficiently general.

Another work that can be compared with our approach is the research of Abadi et al. [**?**]. Like them we propose a logical model to represent and manage the RDF information but their approach is based on the *vertical partitioning* of RDF. Practically the triples table is rewritten into n two-column tables where n is the number of unique properties in the data. In these tables, the first column contains the subjects that define that property and the second column contains the object values for those subjects. In Figure 4.2 the tables of vertical partitioning representation for the example of Section 2 are depicted. Therefore, their approach is based on the "semantic" of the properties. On the other side our approach is based on the particular meta-constructs used to

[3]KAON - the Karlsruhe Ontology and Semantic Web Tool Suite. http://kaon.semanticweb.org/.

[4]TAP project, 2002. http://tap.stanford.edu/.

Type

URI	type
URI1	Person
URI2	Person
URI3	Person
URI4	Person

Name

URI	name
URI1	Priam
URI2	Hector
URI3	Astyanax
URI4	Paris

Child

URIFrom	URITo
URI1	URI2
URI1	URI4
URI2	URI3

Brother

URIFrom	URITo
URI2	URI4

Figure 4.2: The *vertical partitioning* tables for the running example.

represent the RDF elements.

Referring to the example of Section 2, we can formulate the SQL query to find all the name of the persons that have a child whose name is '*Paris*' as follows:

```
SELECT N.obj FROM Name AS N, Name AS N2, Child AS C WHERE
N.subj=C.subj AND C.obj=N2.subj
      AND N2.obj="Paris"
```

As we can see the number self-joins are significantly reduced. However, et us analyze the second and more complex query (i.e. to find the family relation between *Astyanax* and *Hector*). Using the vertical partitioning approach, we must firstly extract the tables list (because each table represents a property), insert them in a view and perform the joins with this view. In this case the query becomes more complex (and we omit it for the sake of brevity) and the total number of joins will significantly increase.

To overcome some of the aforementioned problems, we develop an approach which main characteristics can be summarized in two features: strong flexibility and extendibility. The proposed approach represents concepts of RDF pointing out semantics of elements. We also take into account both blank nodes and RDF containers (i.e. `Bag`, `Seq` and `Alt`) In this way information stored in an RDF document are properly grouped making effective the querying process.

4.4 Management of RDF data

Overview

An RDF model is comparable to a directed labeled graph. However, it allows the presence of multiple edges between two nodes, and different edges between two nodes can share the same label. The nodes, representing resources, can be classified as URI references or blank nodes or literals (strings). The edges in the graph represent properties.

Formally, we can define the following sets: the set of resources R, the set of URI references U, the set of blank nodes B, the set of literals L, and the set of properties P. At RDF level these sets present the following properties: $R = U \cup B$, $P \subset R$, and U, B, and L are pair-wise disjoint. In P the property `rdf:type` defines the type of a particular resource instance: any resource can be the target of an `rdf:type` property. Therefore an RDF model M is a finite set of triples (i.e. statements) as

$M \subset R \times U \times (R \cup L)$.

Each triple or statement in an RDF model contains a resource, an URI reference (which stands for a property), and a resource or literal. The properties in an RDF model are the middle element of a triple, or they are a resource with an `rdf:type` property to the `rdf:Property` resource. So the set of properties of an RDF model M is

$P = \{p | (s, p, o) \in M \vee (p, \texttt{rdf:type}, \texttt{rdf:Property}) \in M\}$

The graph model G_{RDF} corresponding to an RDF model M is

$G_{RDF} = (N_{RDF}, E_{RDF}, fl_{N_{RDF}}, fl_{E_{RDF}})$

$fl_{N_{RDF}} = N_{RDF} \rightarrow R \cup L$

$fl_{E_{RDF}} = E_{RDF} \rightarrow P$

using the following construction mechanism (N_{RDF} and E_{RDF} denote nodes and edges respectively, $fl_{N_{RDF}}$ and $fl_{E_{RDF}}$ their labels). For each $(s, p, o) \in M$, we add the nodes n_s, n_o to N_{RDF} (different only if $s \neq o$) and label them as $fl_{N_{RDF}}(n_s) = s$, $fl_{N_{RDF}}(n_o) = o$, and add e_p to E_{RDF} as a directed edge between n_s and n_o and label that as $fl_{E_{RDF}}(e_p) = p$. In the case that s and/or o are in B, then $fl_{N_{RDF}}(n_s)$ and/or $fl_{N_{RDF}}(n_o)$ are not defined: blank nodes do not have labels. Nodes that have a label have a unique one, edges always have a label but can share it with other edges.

Although an RDF model has the advantage of being general and simple, it cannot be used as a storage model as it is since even simple management operations involve serious performance limitations.

The management of a relevant amount of RDF data requires, in our ap-

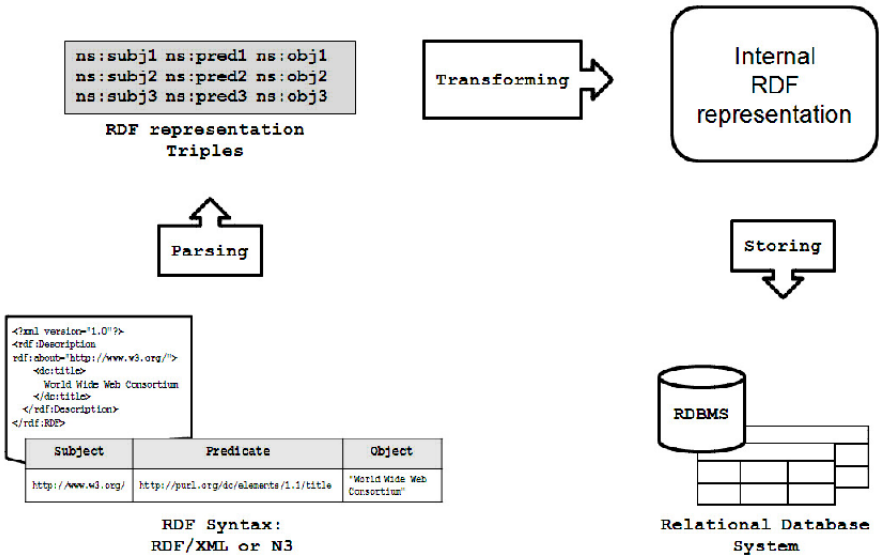


Figure 4.3: RDF data Management with a 3-layer model

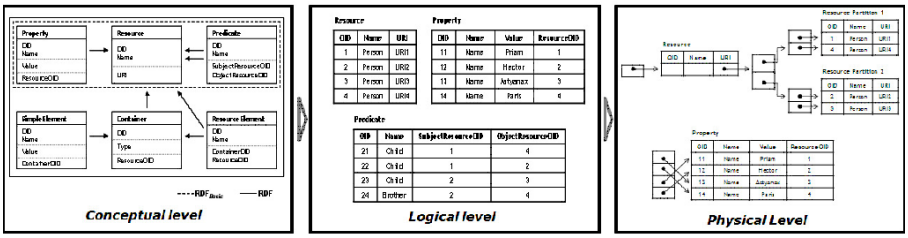


Figure 4.4: The three levels of abstraction

proach, a specific storing process, as shown in Figure 4.3.

This process can be characterized by three steps:

- The first step of the process is the parsing of RDF documents. We consider RDF data represented with different syntaxes (e.g. RDF/XML and N3). As a result of this step we obtain a triple representation of the documents (i.e. $\langle s, p, o \rangle$ statements).

- In the second step the system maps the triple representation into an internal organization. It presents
  - a *conceptual level*, proposing a simple conceptual model where a set of constructs properly represents RDF concepts. Each construct is used to properly represent elements of RDF documents, with the same semantics;
  - a *logical level*, implementing our conceptual model into a logical one. In our case we use the relational model;
  - a *physical level*, defining the physical design of the logical representation of previous level. There, we illustrate a special partitioning technique that relevantly increases the performance of the entire process.
- In the last step the storing in a specific DBMS is performed. In our case we use a relational DBMS.

In the following we exploit our organization of RDF data, from the conceptual to the physical organization, through the logical one. We close the section highlighting the advantages of such data organizations.

Conceptual level

Our approach is inspired by works of Atzeni et al. [**?**, **?**] that propose a framework for the management of heterogeneous data models in an uniform way. They leverage on the concept of metamodel that allows a high level description of models by means of a generic set of constructs. The various constructs have an identifier, are related each other by means of mandatory references and may have attributes that specify details of interest. Formally, a model can be represented as $M = \{C_1, C_2, \ldots, C_n\}$ where the C_i's are constructs, that have the following structure:

$C = (OID, attr_1, \ldots, attr_f, ref_1, \ldots, ref_m)$

where OID is the object identifier, the $attr_j$'s are the properties of the construct and the ref_k's are the references of the construct.

Following this idea, we propose a simple model where a set of constructs properly represent concepts expressible with RDF. The main differences with respect to the approach of Atzeni et al. are:

- they consider a set of constructs able to represents several data models, while we consider a reduced set of constructs because we use those constructs to represent just the RDF model;
- they consider a well marked distinction between schema and instance level. Constructs and rules for the instance level are automatically derived from structures at schema level. For our purposes, we need to manage only instances, since RDF documents contain essentially data. RDFS is devoted to specify schemas for RDF documents, but the modeling of such standard is beyond the focus of this work.

In order to represent the basic concept of a triple, two constructs should be enough, one to represent resources and one to represent statements involving two resources. The concept of resource is clearly defined and can't be subsequently specified, while the concept of statement allows for a more precise modeling. In fact, we can distinguish between two kinds of statement on the basis of its object, that can be a literal with a primitive type value or a resource with its own URI. We named these constructs *Property* and *Predicate*, respectively. More in details the *Resource* construct has a *URI* attribute to store the URI of the resource (we use the internal OID representation as URI for a blank node); the *Property* construct has a reference to the *Resource* it belongs to and has a *value* attribute to store the value of the object; the *Predicate* construct has two *Resource* references to represent the subject and the object of a statement.

Formally we define

$$M_{RDFbasic} = \{C_{Resource}, C_{Property}, C_{Predicate}\}$$

where the constructs have the following structure:

$$\begin{aligned} C_{Resource} &= (OID, URI) \\ C_{Property} &= (OID, Name, Value, ResourceOID) \\ C_{Predicate} &= (OID, Name, SubjectResourceOID, \\ &\quad ObjectResourceOID) \end{aligned}$$

Referring to an RDF Model M and its corresponding graph representation G_{RDF}, we have the following correspondences with our conceptual model M_{RDF}

$\{R \cup U \cup B\} \mapsto C_{Resource}$

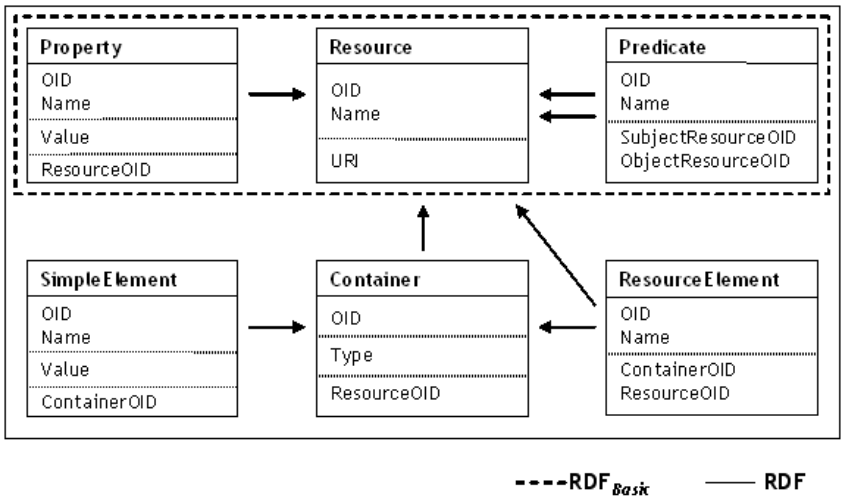


Figure 4.5: The M_{RDF} model

$\{p \in P | (s, p, o)\ where\ s, o \in R\} \mapsto C_{Predicate}$
$(\{p \in P | (s, p, o)\ where\ o \in L\}, L) \mapsto C_{Property}$

where $A \mapsto C_i$ means that the instances of C_i represent the elements of the set A. Moreover the labels $fl_{N_{RDF}}$ and $fl_{E_{RDF}}$ correspond respectively to the attributes *URI* of $C_{Resource}$ and *Name* of $C_{Predicate}$ and $C_{Property}$.

This simple model can be extended in order to manage also RDF collections; this is done with the addiction of three new constructs. They are *Container*, *SimpleElement* and *ResourceElement*. We use the first one to specify that a blank node (represented with a resource as well) is in practice a collection: this is done by means of the reference *ResourceOID* of the construct toward a resource. The construct *Container* has also a property *Type* to denote the type of collection (i.e. `Seq`, `Alt`, `Bag`). The others constructs represent elements of a collection that can be, respectively, literals and resources; a literal element has a *Value* attribute to store the value of the object while a resource element has a reference toward a resource (to specify to which resource the elements of the collection belong to) and both have a reference to the container to which they belong to. Formally we define

$$M_{RDF} = \{C_{Resource}, C_{Property}, C_{Predicate}, C_{Container},$$

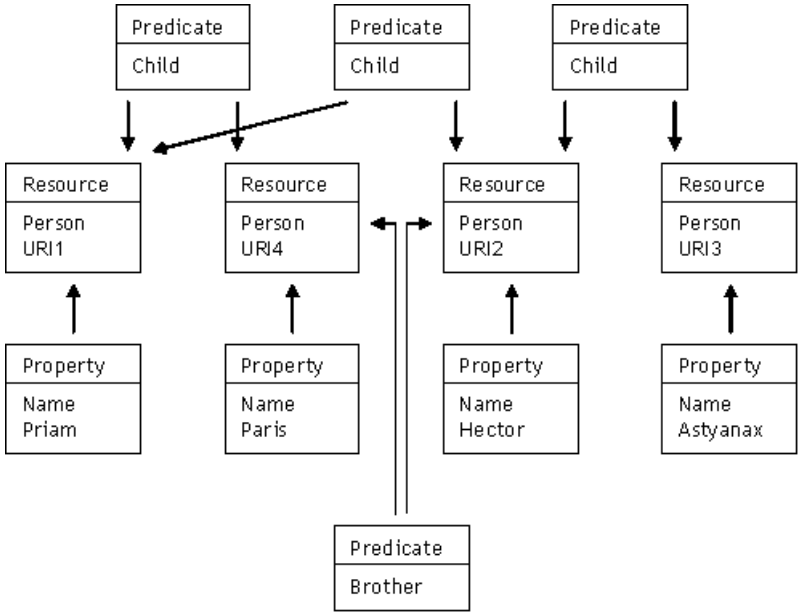


Figure 4.6: Conceptual representation of the running example

$$C_{SimpleElement}, C_{ResourceElement}\}$$

where the new constructs have the following structure:

$$\begin{aligned} C_{Container} &= (OID, Type, ResourceOID) \\ C_{SimpleElement} &= (OID, Name, Value, \\ &\quad ContainerOID) \\ C_{ResourceElement} &= (OID, Name, ResourceOID, \\ &\quad ContainerOID) \end{aligned}$$

In Figure 4.5 an UML diagram of our M_{RDF} model is represented, where enclosed in the dashed box there is the $M_{RDFbasic}$ model.

For instance, through our approach, we can represent the RDF document, presented in Section 2, as depicted in Figure 4.6, where we omit the OIDs for the sake of simplicicty and represent references only by arrows. In the

Resource

OID	Name	URI
1	Person	URI1
2	Person	URI2
3	Person	URI3
4	Person	URI4

Property

OID	Name	Value	ResourceOID
11	Name	Priam	1
12	Name	Hector	2
13	Name	Astyanax	3
14	Name	Paris	4

Predicate

OID	Name	SubjectResourceOID	ObjectResourceOID
21	Child	1	4
22	Child	1	2
23	Child	2	3
24	Brother	2	4

Figure 4.7: Logical representation of our model

figure *URI1*, *URI2*, *URI3* and *URI4* represent the URIs of the instances of the resources *Person*.

Logical Level

We use a relational implementation of our conceptual model. For each construct we create a table and for each field of a construct we add a column to the table corresponding to such construct.

Then we add an integrity constraint for each reference, from the pointer construct to the pointed one (i.e. from the column corresponding to the reference field toward the OID column of the referenced construct). In Figure 4.7 some tables of our logical organization are depicted (we omit the tables devoted to represent collections for the sake of simplicity, because there are no collections in our running example). The value of the rows are those corresponding to our running example.

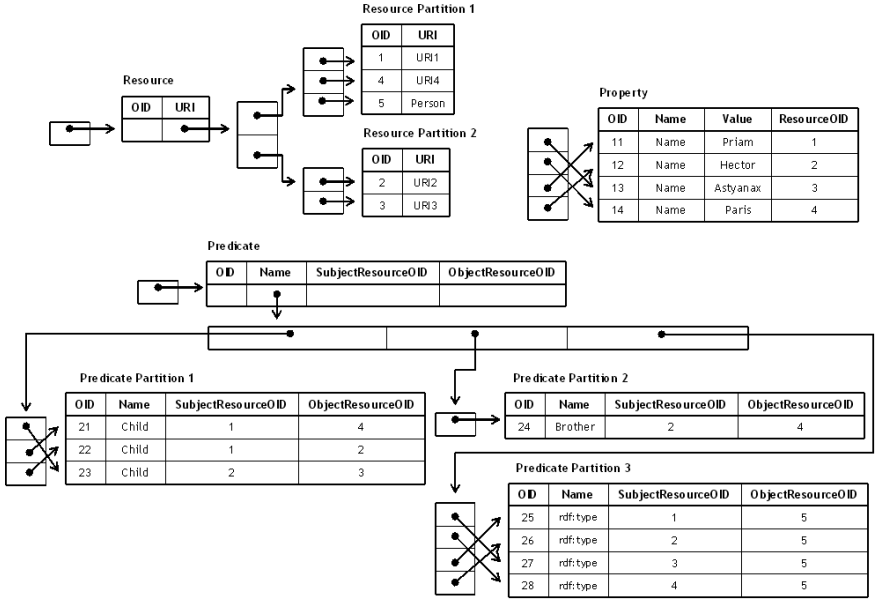


Figure 4.8: Physical Organization of the running example

Physical level

The resulting tables of logical level could be very large. To this aim we use a partitioning technique referring to splitting what is logically one large table into smaller physical pieces.

The partitioning is based on table *inheritance*. Each partition represents a child table of a single parent table. The parent table itself is normally empty; it exists just to represent the entire data set. The child table inherits the structure of the parent (i.e. attributes). The partitioning of a table is processed by the *range* defined on it. In other words the table is divided into different partitions defined by a key column or set of columns, with no overlap between the ranges of values assigned to different partitions.

In detail we set up a partitioned table by following steps:

1. We create the parent table, from which all of the partitions will inherit.

2. The parent table will contain no data. We can define several constraints

on this table (e.g. key, foreign key and so on) to be applied equally to all partitions.

3. We choose the range from the parent table (a single attribute or a set). Then based on this range, we create several child tables that each inherit from the parent table. The child tables inherit the structure of the parent(i.e. attribute and constraints). Normally, these tables will not add any columns to the set inherited from the parent. Though they are partitions of the parent tables, in every way they are normal tables.

4. We add table *check* constraints to the child tables to define the allowed key values in each partition.

5. We create an index on the key column(s) in each partition (as well as any other indexes we might want).

6. Finally we define a trigger or rule to redirect data inserted into the parent table to the appropriate partition.

7. Optionally we can iterate the partitioning on the resulting child tables.

In our case we defined the following ranges:

- $C_{Predicate}$: the key column is the $Name$ attribute. It redirects data into respective partition respect to a unique value. Then we can create indexes on the $SubjectResourceOID$ and $ObjectResourceOID$ attributes.

- $C_{Property}$: the key column is the $Name$ attribute (as well as for $C_{Predicate}$) and we can create indexes on the $ResourceOID$ and $Value$ attributes.

- $C_{Resource}$. the key column is the URI attribute. It redirects data into respective partition respect to a range of values (e.g. alphabetical ranges).

Let's consider the example depicted in Figure 4.8. It illustrates the physical design of the running example, described in Section 2. For instance the $C_{Predicate}$ table was partitioned into two tables respect to the two key values of $Name$ as $Child$ and $Brother$. Also the $C_{Resource}$ table was partitioned into two tables respect to the range values of URI (in this case respect to alphabetical ranges $[a-h]$ and $[p-z]$).

Remarks

We have described our idea of modeling RDF data at different levels: conceptual, logical and physical level. Each level provides relevant benefits. The novelty introduced at conceptual level is a characterization of RDF elements. Whatever is the syntax used in an RDF documents, all the elements are represented in a uniform way. We distinguish between resources and triples; then we characterize the triples on the basis of their object and we use a specific representation to address RDF containers and their elements. At logical level, the choice of a relational implementation grants us all the advantages of the most used model. In particular, it is possible to use several specific tools and SQL querying facilities. At physical level, partitioning gives serious advantages. The benefits will normally be worthwhile only when a table would otherwise be very large. Let's consider query performance that can be improved dramatically in certain situations. In particular partitioning helps when most of the heavily accessed rows of a table are in a single partition (or a small number of partitions). Also update operations (e.g. deletion) exploits the partition accessing only the corresponding portion of table to update. Normally the set of partitions established when initially defining the tables are not intended to remain static. It is common to want to remove old partitions of data and periodically add new partitions for new data. One of the most important advantages of partitioning is precisely that it allows this otherwise painful task to be executed nearly instantaneously by manipulating the partition structure, rather than physically moving large amounts of data around. This kind of partition can reduce the number of the joins and also reduces the number of unions that otherwise would make much more complex the query process.

Moreover the three level we consider define also the typology of user that can access to the system, from less expert (i.e. the conceptual level) to more expert (i.e physical level).

Other advantages of the proposed idea are listed in the following.

In RDF documents a property/predicate may appear more than once with the same subject resource but different object resource (this is called multi-valued attribute). The management of multi-valued attributes is easily implemented in our approach. At conceptual level, each object value corresponds to a new instance of the property/predicate construct. At logical level, a new instance of a construct corresponds to the addiction of a row in the construct table.

In our approach, as stated in previous sections, blank nodes are managed like other resources. A blank node differs from a common resource in the value

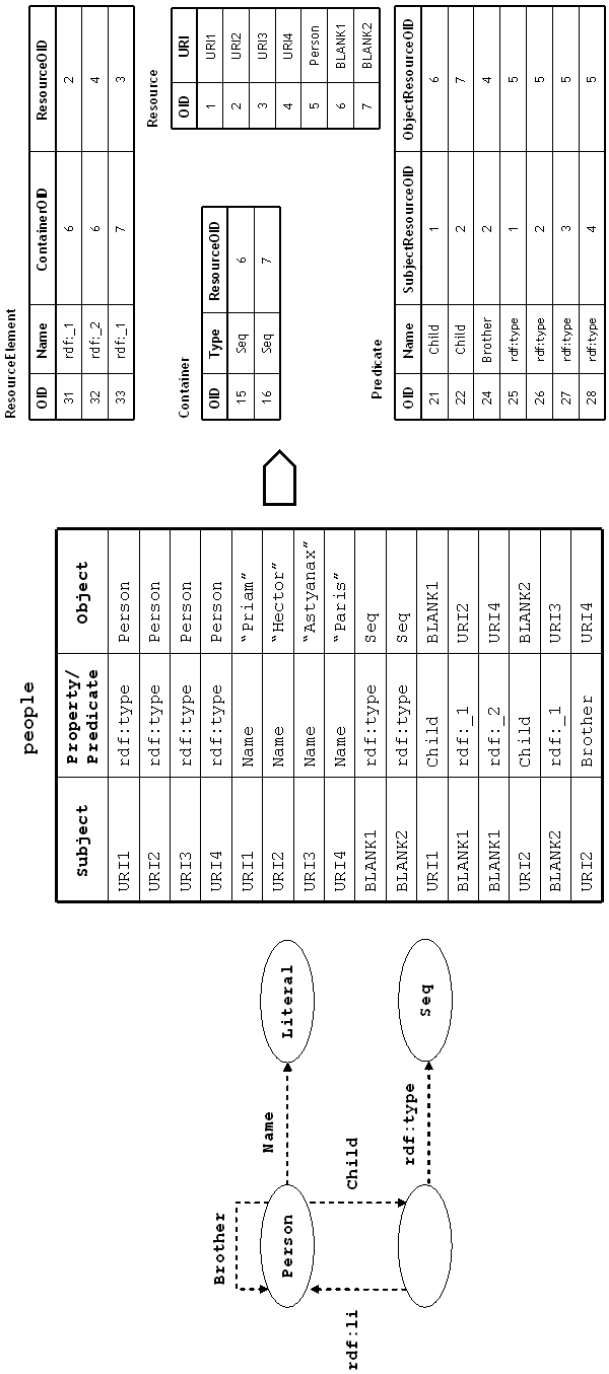


Figure 4.9: Logical representation of Containers

of the URI attribute. The blank node has a fake URI equal to the internal OID representation.

Let us consider a variant of the example of Section 2, where we represent the children of a person with an RDF container (namely a `Seq` container). This situation is depicted in Figure 4.9(a). With our model, we can address RDF collections exploiting the constructs `Container`, `SimpleElement` and `ResourceElement`. In particular, we represent a collection with an instance of `Resource` construct (a blank node) and each element of the collection with an instance of `SimpleElement` or `ResourceElement` construct (depending on the type of elements). An instance of `Predicate` construct links the container with the proper resource. The logical representation is presented in Figure 4.9(b) (where we omit not relevant constructs). We define a new instance of a construct for each element of an RDF document. Therefore every construct instance always has a value for each field (i.e. OID, properties and eventually references).

Like Abadi et al., we have a fixed structure that does not require any kind of clustering. In other words, we can refine our model, adding new constructs, but there is not the need to change already defined constructs (i.e. the number of fields of a construct will not change).

4.5 Experimental Results

In this section, plenty of experiments have been done to evaluate the performance of our framework. We illustrate the RDF Benchmark chosen (i.e. a public available dataset and a set of seven representative queries). Then we compare performance and scalability of our approach with Triple and Vertical Partitioning. We will call our approach *Semantic Web Information Management* (SWIM).

RDF Benchmark

We used the public available Barton Libraries dataset[5], provided by the Simile Project[6]. This dataset is a collection of RDF documents (i.e. around 11.000) containing records (formatted according to the RDF data model specifications) acquired from a dump of the MIT Libraries Barton catalog. This collection of files was derived from diverse sources. Therefore the structure of the data

[5]Library catalog data. `http://simile.mit.edu/rdf-test-data/barton/`
[6]Simile website. `http://simile.mit.edu/`

is quite irregular and often presents RDF malformed URIs. We converted the Barton dataset from RDF/XML syntax to triples using the Jena parser and then we made a soft cleaning of data (i.e eliminating duplicate triples and malformed URIs). The parser phase resulted a total of 60.578.683 triples. As documented by Abadi et al. [**?**], the dataset contains 260 predicates and 23 properties. Many of them are multi-valued: they appear more than once for a given subject. Respect to Abadi et al. we maintained triples with particularly long literal values (as the property *abstract* in the dataset) and triples with properties or predicate with few occurrences (as the predicate *_1953*). This is because we want to prove our experiments with the real unstructured nature of Semantic Web data.

Our experiments want to compare the SWIM methodology with the common Triple storage technique and in particular with the Vertical Partitioning approach, which revealed itself one of the most performing approach. Therefore we implemented the seven representative queries used by Abadi et al. The full queries are briefly described at a high level here. We illustrate the complexity of each one and later we will discuss how these queries are executed by the different approaches.

Query 1 (Q1) This query counts the number of different resources, object of the predicate *type*. This requires a scan for the triples with predicate *type* and a counting for the different objects of these triples. In this case, in each triple, the object resource is the variable to be processed.

Query 2 (Q2) This query counts the occurrences of all properties or predicates coming out from resources that are subject of triples with predicate *type* and object a resource named *Text*. In this case the property or predicate for each triple is the variable to be processed .

Query 3 (Q3) Following the query Q2, Q3 counts the occurrences of all properties or predicates (and corresponding objects) coming out from resources, subject of triples with predicate *type* and object a resource named *Text*. The complexity of this query is similar to the previous.

Query 4 (Q4) Starting from the counting of all properties (predicates)-object from Q3, this query recalculates these counts where the subject of triples with predicate *type* and object a resource named *Text* has also an outcoming property *language* with value *fre* (i.e. French). This query is thus similar to

Q3, but has a higher level of result selectivity comporting a more larger space of searching.

Query 5 (Q5) Here the query results a subject s and an object o such that s has an outcoming predicate *origin* with object a resource named *DLC* and an outcoming predicate *records* having an object that is also subject of a triple with predicate *type* and object o. Also this query has a high level of selectivity where resources are the variables to process.

Query 6 (Q6) This query combines Q2 and Q5. It returns all the predicates or properties coming out from a resource or subject of a triple with predicate *type* and object a resource named *Text* or subject of a triple with predicate *records* having an object that is also subject of a triple with predicate *type* and object *Text*. This query presents a relevant complexity due to the high selectivity and huge amount of data to process in the dataset. Also in this case in each triple predicate and properties are variables to process.

Query 7 (Q7) This final query presents a selection of three types of resources, respectively r_1, r_2 and r_3 such that r_1 is the subject as of a triple with predicate *point* and object a resource named *end* as of a triple with predicate *type* and object r_2 as of a triple with predicate *encoding* and object r_3. In this case resources are variables to process.

Platform Environment

Our benchmarcking system is a dual quad core 2.66GHz Intel Xeon, running Debian, with 8 Gbytes of memory, 6 MB cache, and a 2-disk 1Tbyte striped RAID array.

We implemented our experiments using Postgres. As Beckmann et al. [**?**] experimentally showed, it is relevantly more efficient respect with commercial database products.

Our Postgres implementation of the Triple storage provides a table containing three columns: *subject*, *property* and *object*. We defined a primary key on the columns triple (subject, property, object) and used B+ tree indices. One clustered on (subject, property, object), three unclustered respectively on subject, predicate and object[7].

[7]These indexes were experimentally determined to achieve the best performing results

We implemented the Vertical Partitioning storage using one table per property. Each table contains two columns: *subject* and *object*. The primary key is the columns couple (subject, object). We used two indexes: a clustered B+ tree index on subject, and an unclustered B+ tree index on object.

The SWIM storage exploits the native partitioning technique of Postgres. We used three tables for *Resource*, *Predicate* and *Property* with the relational schema described in Section 4.3. They represent the parent tables partitioned respect to the ranges described in the Section 4.4. On the Resource table (and relative partitions), we used a constraint of unique value on the *URI* attribute. On the Property and Predicate tables we used an unclustered B+ tree index on the *Name* attribute and for each partitions we used clustered B+ tree indexes on the *SubjectResourceOID* and *ResourceOID* attributes, and unclustered B+ tree indexes on the *ObjectResourceOID* and *Value* attributes. Each partition has a check constraint and a rule to redirect data inserted into the parent table.

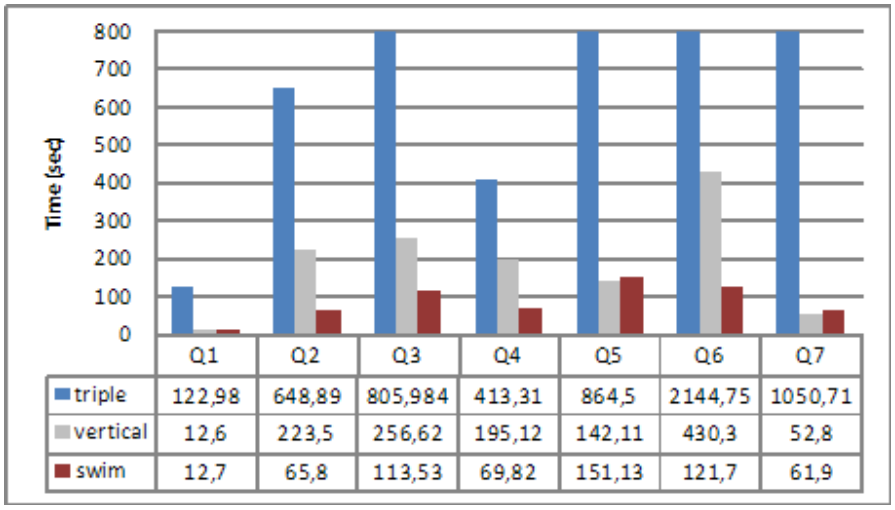
Evaluating Results

Performance Results The seven queries have been tested on the three implementations and the resulting query execution times are shown in Figure x. Each number is the average of three runs of the query. Internal database cache and operating system cache can cause false measurements then before each query execution we have restarted the database and cleaned any cache.

In the following we will explain the obtained results for the execution of each query.

Q1 The Triple approach presents a relevant response time, compared to the other approaches, due to several self-joins needed. Vertical Partitioning and SWIM, instead, have comparable results. In the Vertical approach only the *type* table is accessed, performing a count operation on the different values of the *object* attribute. To answer this query, SWIM has to access the join between the *predicate* and *resource* tables. Not the whole tables are involved in the join operation thanks to the partitioning technique. It automatically redirects to the partitions corresponding to the *type* predicate and the *Text* resource.

Q2 The Triple approach needs to extract all the subjects from the triple with property *type* and object *Text*. Then a self-join on the subject is performed to extract all other properties and count them.



	Q1	Q2	Q3	Q4	Q5	Q6	Q7
triple	122,98	648,89	805,984	413,31	864,5	2144,75	1050,71
vertical	12,6	223,5	256,62	195,12	142,11	430,3	52,8
swim	12,7	65,8	113,53	69,82	151,13	121,7	61,9

Figure 4.10: Performance comparison between triple-vertical-swim approaches

The Vertical accesses the table *type* selecting the subject for all the tuples with object *Text*. Then a scan for all the other tables is performed to join on the subject and count. This is a typical case where the property is unknown then the Vertical Partitioning needs to scan all the database tables that refer to properties. This is due to the fact that the name of the table is the property, therefore a query to the database metadata is needed. Finally all the results are collected by a union operation.

The SWIM approach queries the predicate table and, relying the partitioning mechanism, accesses directly the *type* partition to select the subjects for the object *Text*. Then the previous result is used in a nested query to extract all the properties and predicates with the corresponding subjects. The partitioning techniques avoid to scan the whole tables (millions of tuples) and access only the portions of interest. Moreover we avoid the huge amount of unions executed by the Vertical.

Q3 Since the query Q3 is similar to Q2, we don't illustrate all the details. In this case the Vertical approach presents similar response time. The additional operation on the objects costs a 25% more to the Triple approach. The execution time of SWIM increases consistently due to the necessity of join with the

resource table. However, also in this case, the partitioning mechanism supports the performance of this operation.

Q4 Respect to Q3, Q4 introduces a higher level of selectivity. This allows the tree approaches to exploit the indexing mechanisms and reduce the response time. We benefit more than the others exploiting the combination between indexing and partitioning.

Q5 In this case, Triple needs to perform three heavy self-joins. Since the properties are known, Vertical obtains the best result. It directly accesses the specific tables selecting subjects and objects. SWIM presents a comparable time but higher due to the join operations with the resource table.

Q6 This query has the highest complexity due to the explicit union operation between the results of two subqueries. Triple dramatically suffers in this case. In both subqueries the property is unknown then Vertical needs to apply each to every table and make union of the results. This is the case where SWIM can exploit at best its logical representation combined with the physical optimization.

Q7 Like Q1 and Q5, properties are known and the resources related to the properties should be selected. Again, in this case, Vertical is the most performing due to the aforementioned advantages.

Summarizing, the Triple approach presents the worst results, due to its unprofitable storage model. Vertical obtains best results in the queries where the properties are known (i.e. Q1, Q5, Q7) because the corresponding tables are accessed directly. The results of SWIM are better in the other cases due to its internal data organization (conceptual, logical, physical). However, in Q1, Q5, Q7 SWIM response times are comparable with Vertical.

Scalability Results To measure the scalability of each approach, we have performed an incremental import. The Barton file have been divided into small subsets of triples and subsets are imported separately. The import of a subset, corresponds to adding new knowledge to the RDF graph. We have measured the time needed to insert the new triples in each approach... After the import of each subset, the queries are executed for the three approaches measuring the query answer time. Figure X

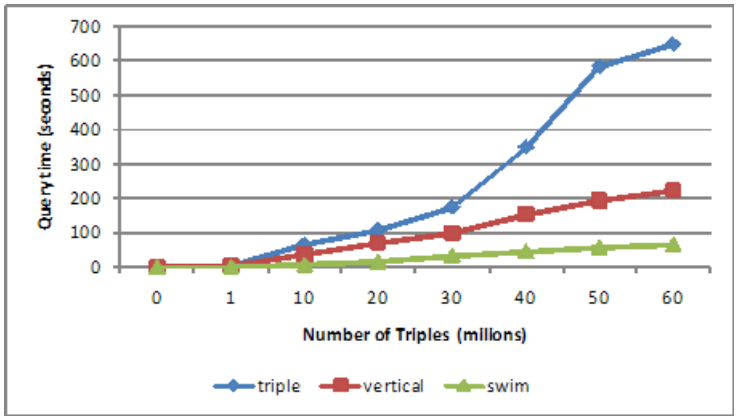


Figure 4.11: Query Performance as number of triples scale

Although the magnitude of query performance is important, an arguably more important factor to consider is how performance scales with size of data. In order to determine this, we varied the number of triples we used from the library dataset from one million to fifty million (we randomly chose what triples to use from a uniform distribution) and reran the benchmark queries. Figure 4 shows the results of this experiment for query 6. Both vertical partitioning schemes (Postgres and C-Store) scale linearly, while the triple-store scales super-linearly. This is because all joins for this query are linear for the vertically partitioned schemes (either merge joins for the subject-subject joins, or index scan merge joins for the subject-object inference step); however the triple-store sorts the intermediate results after performing the three selections and before performing the merge join. We observed similar results for all queries except queries 1, 4, and 7 (where the triple-store also scales linearly, but with a much higher slope relative to the vertically partitioned schemes).

4.6 Conclusions

Due to the growing importance of Semantic Web, a number of applications that uses RDF data has been developed. This large amount of data must be rapidly accessed in order to be effectively used. Storing and maintaining RDF data represent a crucial activity to achieve this complex purpose. The classical "triple-stores" approaches are not good enough because most of the

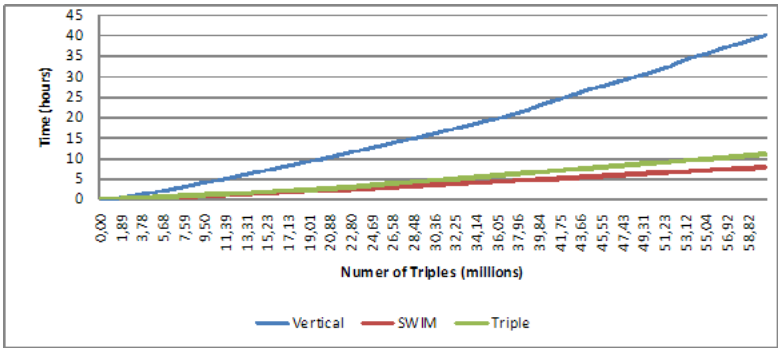


Figure 4.12: Maintenance Performance as number of triples scale

queries require a high number of self-joins on the triples table. In order to overcome these problems we proposed a model-based approach to store and maintain large amount of RDF data and showed that it achieves performance results similar to other well-known approaches. Moreover, the scalability tests performed have shown the quality of the overall approach.

The expressive power of the proposed model depends on the number of constructs we have introduced. If the need to represent new concepts arises, we have just to add proper constructs to the model. In particular, the goal of our current work involves the management of both RDF (data level) and RDFS (schema level) via an extended model. Another issue we are studying regards the possibility to integrate the management of OWL from a data management perspective.

Chapter 5

Living with ontologies and databases

Interoperability of ontologies and databases has received a lot of attention recently. However, most of the work has concentrated on specific problems (such as storing an ontology in a database or making database data available to ontologies) and referred to specific models for each of the two. Here we propose an approach that aims at being more general and model independent. In fact, it works for different dialects for ontologies and for various data models for databases. Also, it supports translations in both directions (ontologies to databases and vice versa) and it allows for flexibility in the translations, so that customization is possible. The proposal extends recent work for schema and data translation (the MIDST project, which implements the ModelGen operator proposed in model management), which relies on a metamodel approach, where data models and variations thereof are described in a common framework and translations are built as compositions of elementary ones.

5.1 Introduction

The availability of mature Semantic Web technologies and the definition of standards encourage the development of applications that exploit this approach. While Semantic Web formalisms are conceived to define ontologies or simple annotations with the purpose to share and reuse knowledge [**?**], databases are built to store (and retrieve) information effectively and efficiently. Even if the prerequisites and the general goals are rather different,

Semantic Web and (relational) database applications coexist with a growing need of interoperability. Large ontologies are often stored in database repositories in order to exploit their ability to handle secondary storage and to answer queries in efficient ways. Semantic Web query languages can also be translated to SQL thus to allow Semantic Web applications to keep on using their own query languages over data that is both internal and external. Moreover, the visibility of data stored in databases can be extended to Semantic Web applications, automatically generating a new ontology from the source data or mapping to an existing ontology. Another interesting aspect about Semantic Web and Database interoperability regards Web sites. Nowadays many Web sites are dynamically generated from a database, therefore it would be useful to let this content be available for Semantic Web applications [**?**, **?**].

Many forms of mappings between ontologies and database have been proposed in the literature. Most of them use an OWL ontology to describe the database structure and data. The rules to implement the correspondences are often embedded in the tool. Moreover, to the best of our knowledge, each work focuses either on the translation from ontologies to databases or on the reverse one, but does not covers both at the same time.

We propose here a completely different approach that allows the translation between Semantic Web formalisms and databases and backwards. Moreover, we aim at supporting different models both for ontologies and for databases. We pursue this goal by adopting a technique based on a *metamodel* approach, as proposed in MIDST (Model Independent Schema and Data Translation) [**?**, **?**]. The metamodel is the set of constructs that are available, and a model is defined by indicating the constructs (and their variations) it includes. The metamodel we need here is an extension of the one used in MIDST, with many common constructs and a few additional ones, needed to discuss features that appear in ontologies and not in traditional database models. In this framework, we have a special model, called the *supermodel*, that includes all possible constructs, and so generalizes all models of interest. A major benefit of the supermodel is the fact that all translations of interest can be performed within it, by means of compositions of elementary steps that refer to the specific features of the various constructs. Such a framework allows for the specification of many different models and for the support to the translation between each of them. Translation rules are defined in a Datalog variant with OID invention, and so they are independent of the engine that executes them and also easily customizable and extendible. Within this framework we are now conducting various experiments, and we discuss in this thesis how we can handle translations between OWL-Lite ontologies and relational databases (in both directions). We choose

OWL-Lite for these preliminary experiments because it is currently the most commonly used variant of OWL.

The main contributions of this work are:

- The extension of the metamodels and of the supermodel to properly represent Semantic Web formalisms
- The possibility to operate translations between Semantic Web languages and the relational model in both directions
- A flexible framework that allows the user to manipulate translation rules with a powerful language.

The rest of the chapter is organized as follows. In Section 5.2 we review related work in the literature. In Section 5.3 we illustrate the extensions we need for MIDST supermodel in order to deal with Semantic Web features. In Section 5.4 we show how our approach covers translations from ontologies to relational databases and in Section 5.5 we cover the converse. Section 5.6 is devoted to brief final remarks.

5.2 Related Work

Many proposals exist that address interoperability issues between Semantic Web and relational databases. Several branches can be identified, and we briefly comment here on some of them. The ability of generating an ontology from relational data has been faced by [**?**, **?**]. Both approaches describe the database schema and instances by means of a suitable dictionary used in an ontology. Handschuh et al. [**?**] apply a similar approach for annotating data intensive Web sites. In these works, mappings are managed referring to specific ontologies that describe the source relational model. A similar approach is followed by [**?**] for the generation of RDF documents. A more recent approach [**?**] discusses various possibilities for the mapping between relational databases and RDF. Our approach aims at a greater generality: we describe the models of interest by a meta-model, subsequently schema and instances are treated. This allows us to be model independent, with a general approach that is extensible to virtually any model. Moreover, translations are not embedded but specified by means of high level rules and therefore they are customizable according to the different needs.

A number of formal approaches based on description logics exist [**?**, **?**]. These cannot be compared with our work, as we concentrate on structural

aspects, and so we do not refer to reasoning capabilities. However, these pieces of work each refer to one specific data model, ER or relational, whereas our approach applies to many different data models, belonging to many families, including relational, ER, object-oriented and object-relational.

Another related direction of research concerns the storage of ontologies. Since storing ontologies into flat files results in a loss of scalability and performance, recent proposals have considered the adoption of relational databases as repositories. In [**?**] the framework MINERVA is presented with the goal of storing large ontologies into a relational database. The tool achieves scalability and performance issues exploiting a relational schema tailored on the OWL model. Our work addresses the storage of ontologies by exploiting a logical organization of data that is not bound to a specific model, and customization of the translations is possible.

Moreover, none of the above approaches considers translations in both directions, nor any form of model independence, as each of them is tightly related to a specific data model and a specific translation approach.

A related, but different problem is considered in [**?**], who study mappings between databases and ontologies, as a tool to support evolution and capture change. However it provides no support to building the mapping.

5.3 Extending MIDST supermodel to Semantic Web

In this section we propose an extension of MIDST that supports the interoperability of Semantic Web formalism to translate data and schemas from databases to ontologies and viceversa. The supermodel has been defined with respect to a variety of models for which translations can be applied by our approach (E-R, Relational, Object-Oriented, Object-Relational, XSD, etc.). The structure of the supermodel is relatively compact. In our relational implementation, it has a table for each construct. We currently have a plenty of constructs, which are sufficient to describe a large variety of models. However Semantic Web formalisms required a bit more work to be addressed in our approach.

In our preliminary works on definition of Semantic Web models (and consequently the related mapping rules) we have addressed semantic annotation platforms [**?**] and two formalisms, namely RDF and TopicMaps [**?**]. In these works the supermodel has been extended to consider the constructs of the aforementioned contexts.

Despite of its generality, the previous version of the supermodel could not

be used to represent the broad range of formalisms coming from the Semantic Web field. For this purpose, and also to bridge the gap between databases and ontologies, we propose a more general version of the supermodel extending its capabilities.

The supermodel has been redefined to take into account Semantic Web models, maintaining the possibility of using the variety of models so far considered. Therefore we introduce a set of new meta-constructs that can address also ontologies (RDF, RDF(S), OWL, etc.), we also define a set of new rules able to perform the translations between the new models and the old ones. In this thesis we focus on translation between relational databases and ontologies.

A simplified version of the extended supermodel presenting only the constructs of interest is shown in Fig. 5.1. In the diagram for each construct we show the attributes, the properties and the references. In order to better understand the supermodel and its features we briefly introduce the most important constructs with their associated properties (the prefix `SM_` indicates that the constructs belongs to the *supermodel*):

- `SM_Abstract` - Abstract is used to describe all kinds of concepts. It corresponds to constructs used in many models, such as ER model entities and OWL classes. Each object must have an identity to be uniquely identifiable (i.e. the OID).

- `SM_Aggregation` - This construct is used to represent aggregation of lexicals. For example a relational table can be represented as an aggregation of lexicals where each lexical represents one column.

- `SM_BinaryAggregationOfAbstracts` - This construct is used to represent all kinds of binary relation between abstracts. For example they are used to map relationships in ER and `owl:objectproperties` in OWL.

- `SM_ForeignKey` - We use this construct to represent the relation between the *from* table and the *to* table and we use `SM_ComponentOfForeignKey` to describe columns involved in the foreign key.

- `SM_InverseOfBinaryAggOfAbs` - This construct represents the inverse of a `SM_BinaryAggregationOfAbstract`. In particular it can be used to map the inverse of an OWL property.

- `SM_Generalization` - It is used jointly with the construct `SM_ChildOfGeneralization` to represent generalization. For example it is used to map ER generaliza-

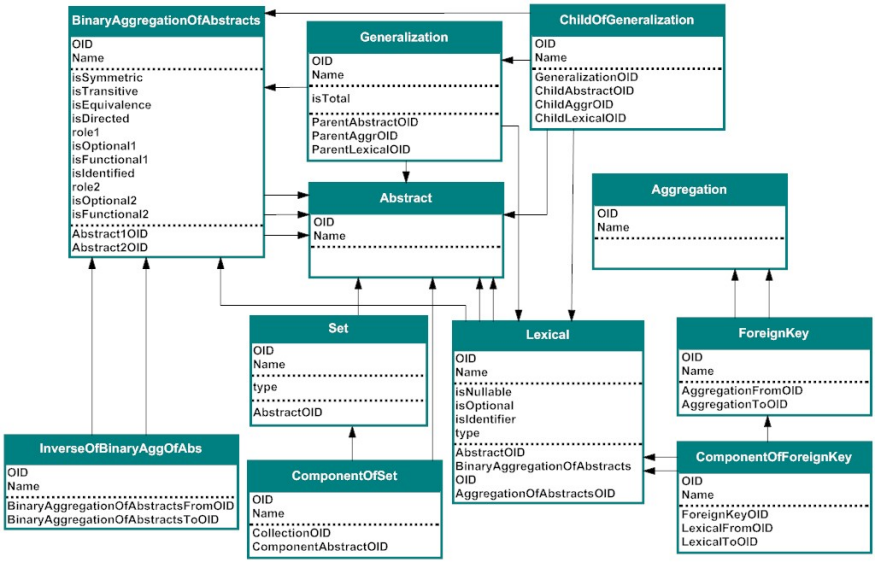


Figure 5.1: A portion of MIDST supermodel.

tions and also to describe the concepts of `subClassOf` and `subPropertyOf` in OWL-Lite.

- `SM_Lexical` - Lexical construct is used to represent each lexical elements, that is an element with an associated value [**?**]. If we are considering the instance level they contains the instance data (such as the individuals of OWL).

- `SM_Set` - This construct is used to represent sets of abstracts. Each element of the set is represented through the `SM_ComponentOfSet` construct. The `type` property specifies which kind of set we are considering. For example we can represent collections such as the RDF containers or even the OWL construct `intersectionOf`.

Each model is defined by its constructs and the meta-constructs they refer to. Simple versions of models could be seen as follows (omitting some details, but keeping the main features):

- the relational model (under a reasonably simplified view) involves (i) tables, that are represented by the meta-construct `Aggregation`, with columns
  (the `ComponentOfAggregation`). Some specification for the columns can be expressed, for example whether they are part of the key or whether nulls are allowed; (ii) foreign keys defined over components of aggregations;

- the OWL-Lite model at a simple level involves (i) the main element are the classes (which correspond to abstracts); (ii) `BinaryAggregationOfAbstracts` that represents the properties (i.e. binary relation between classes).

The proposed extension of the supermodel and the consequent extension of the MIDST tool has permitted to analyze the interoperability issues also for Semantic Web models.

5.4 From OWL ontology to Relational Database

Our purpose is to translate an OWL ontology into a relational representation trying to avoid any information or semantics loss. We will illustrate by a use case how the translation takes place. Let us consider the OWL-Lite example below:

```
:Person a owl:Class . :Surname a owl:FunctionalProperty ,
    <http://.../owl#DatatypeProperty> ;
    rdfs:domain :Person ;
    rdfs:range <http://.../XMLSchema#string> .
:Score a owl:FunctionalProperty ,
    <http://.../owl#DatatypeProperty>;
    rdfs:domain :SoccerPlayer ;
    rdfs:range <http://.../XMLSchema#int> .
:Name a owl:FunctionalProperty ,
    <http://.../owl#DatatypeProperty>;
    rdfs:domain :Person ;
    rdfs:range <http://.../XMLSchema#string> .
:SoccerPlayer a owl:Class ;
    rdfs:subClassOf :Person ;
    owl:equivalentClass
    [ a owl:Class ;
        owl:intersectionOf
        ([ a owl:Restriction ;
            owl:allValuesFrom :SoccerTeam ;
            owl:onProperty :worksIn
          ] [ a owl:Restriction ;
              owl:maxCardinality
              "1"^^<http://.../XMLSchema#int> ;
              owl:onProperty :worksIn
```

```
            ]) ] .
:hasCaptain a owl:FunctionalProperty ,
    <http://.../owl#ObjectProperty> ;
    rdfs:domain :SoccerTeam ;
    rdfs:range :SoccerPlayer ;
    owl:inverseOf :inverse_of_hasCaptain .
:hasPlayed a owl:ObjectProperty ;
    rdfs:domain :Person ;
    rdfs:range :SoccerTeam .
:worksIn a owl:ObjectProperty ;
    rdfs:domain :Person .
:TeamName a owl:DatatypeProperty ;
    rdfs:domain :SoccerTeam ;
    rdfs:range <http://.../XMLSchema#string> .
:inverse_of_hasCaptain a owl:InverseFunctionalProperty ,
    <http://.../owl#ObjectProperty> ;
    rdfs:domain :SoccerPlayer ;
    rdfs:range :SoccerTeam ;
    owl:inverseOf :hasCaptain .
:SoccerTeam a owl:Class ;
    rdfs:subClassOf <http://.../owl#Thing> ;
    rdfs:subClassOf
    [ a owl:Restriction ;
        owl:minCardinality "1"^^<http://.../XMLSchema#int> ;
        owl:onProperty :hasCaptain ] .
```

The serialization syntax chosen is N3 [**?**], that is more compact and readable than RDF/XML. To save more space we have abbreviate the URIs with "...", when the address is too long.

This example represents the relationships between soccer players and soccer teams. A general class `Person` is defined with the property `worksIn`, to express that a person works somewhere. Each person has a `Name` and a `Surname` of type string.
The class `SoccerPlayer` is obtained as the persons that work in one `SoccerTeam`. The ontology also tells us that a soccer player could have past relationships with other teams, by the property `PlayedInThePast`. The functional property `hasCaptain` represents that each soccer team has exactly one captain (as soccer player). Moreover, `hasCaptain` is also inverseFunctional then an `inverseOfhasCaptain` represents that a soccer player is captain of a single team with the same value as `hasCaptain`. Soccer players have a `Score` of type integer. Soccer teams have a `TeamName` of type string.

OWL To Supermodel

Following the translation process in our approach, the first step is to describe the OWL-Lite constructs in terms of the supermodel constructs. By the MIDST tool it is possible to add models and, for each model its constructs,

defining them by means of the meta-constructs. The result of this phase is a meta-representation of the OWL-Lite model that is stored in the relational dictionary. Fig. 5.2 sketches some of the correspondences to explain how the OWL-Lite constructs are mapped into the supermodel.

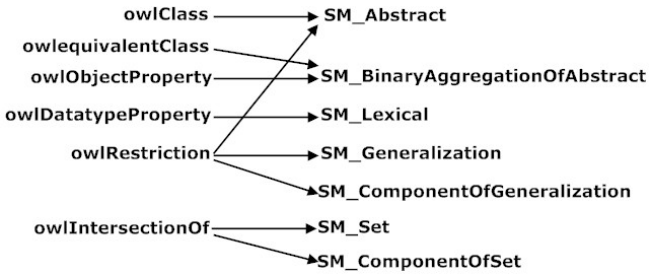


Figure 5.2: Correspondences between the OWL model and the supermodel.

Once the correspondences between the OWL-Lite model and the supermodel are defined, we can exploit them to import an OWL-Lite schema in our relational representation. Due to space limitations, we will not illustrate all the facets of the import process but, referring to the example of section 5.4, we explain some major cases that are representative.

Let us recall, for the sake of completeness, the major feature of OWL we refer to.

Beside the named classes (e.g. `Person`), OWL allows to specify a class as the result of a set operation (equivalence, intersection, etc.). Moreover a class can be defined as all the elements that comply a restriction. A restriction can be on values, when it bounds the range of a property or on cardinality, when it limits the number of values of a property.

Let us now consider the non trivial definition of the class `SoccerPlayer` that is obtained by means of operations on named and unnamed classes. Let C, C_1 and C_2 be unnamed OWL-Classes. Be C_1 the class of objects that comply the restriction on the property `worksIn`, to have all the values from the class SoccerTeam. Be C_2 the class defined as a restriction on the property worksIn, to have 1 as the maximum cardinality. The class C is obtained as the intersection of the classes C_1 and C_2 and represents all the *things* that work in exactly one soccer team. We eventually have all the elements to describe the class `SoccerPlayer`, that is defined as a subclass of `Person` and to be equivalent to the class C, therefore all the persons that work in exactly one soccer team.

Let us illustrate how the elements involved in the `SoccerPlayer` definition are imported into the MIDST tool. The class C_1 is mapped by creating an abstract called `ThingWorksInRestricted`, that is the sub class of the `Thing` class where all the elements works in a soccer team. Then a BinaryAggregationOfAbstracts called `worksInRestricted` is defined, that relates `ThingWorksInRestricted` to the class `SoccerTeam`. At last a generalization can be defined, to espress that `worksInRestricted` is a subproperty of `worksIn`.

For the class C_2, defined as all the things that have at most one value for the property `worksIn`, the mapping process is similar to the previous.

Defining a `Set` of type `intersection`, that refers to the class C, with the classes C_1 and C_2 as `ComponentOfSet`, we can express that C is the intersection of C_1 and C_2. We finally use a binary aggregation of abstract, defined as `owl:equivalentClass`, to relate the class `SoccerPlayer` to the class C. In section 2.4 we have illustrated a portion of the relational dictionary that is used to store all the models information, another part of the dictionary allows to store the schema information. During the importing of an OWL document, for each model construct that appears in the dictionary (the rows of the $MM\_Construct$ table), the system creates a table that is used to manage the schema elements. For example, we have the `owl:Class` construct stored in the $MM\_Construct$ table and the schema element Person stored in the OWL_Class table. A small example that illustrates the hierarchy of some tables used is illustrated Fig. 5.3.

After the first "copy" operation, we have obtained database tables that fully describe the ontology structure, exploiting a logical organization that reflects the constructs of the ontology language. Once the ontology is translated in terms of a relational representation, it can be queried, modified and converted back to the source ontology language. Since all the characteristics of the constructs used to define the source ontology are stored into the relational representation, it is possible to perform the reverse transformation from the relational representation back to the original ontology. Moreover, our meta-representation of an OWL ontology can subsequently be used to perform translations to other formalisms.

Translation within the Supermodel

As a result of the copy operation, we have achieved the goal to store the structural aspects of an ontology into a relational database avoiding semantic loss.

By means of rules, written in a Datalog variant, it is also possible to define

Model

OID	NAME
0	XSD
1	ObjectOriented
2	EntityRelationship
88	Relational
90	OWL

MM_Construct

OID	MODELOID	NAME
215	88	Relation
216	88	Attribute
222	90	Class
223	90	ObjectProperty
224	90	DataTypeProperty

Schema

OID	NAME	MODELOID
6	ER_Invoices	2
7	OO_Invoices	1
10	Rel_PersonDept	88
11	OWL_PersonDept	90

OWL_Class

OID	SCHEMEOID	NAME
7	11	SoccerPlayer
8	11	SoccerTeam
9	11	Thing

OWL_DataTypeProperty

OID	SCHEMEOID	NAME	ABSTRACTOID
1	11	Name	7
2	11	Surname	7
3	11	TeamName	8

OWL_ObjectProperty

OID	SCHEMEOID	NAME	FUNCTIONAL...	ABSTRACT1OID	ABSTRACT2OID
6	11	worksin	true	7	8
7	11	inverseOf_worksIn	true	8	7

Figure 5.3: MIDST Tables.

the translation between the source ontology language and a specific target relational model, chosen by the user, rather than using our meta-representation.

The flexibility of our approach allows the user to choose the form of the translation, defining how to generate the target model. For example, it is possible to define a translation where a relation is generated for each kind of class and another translation where only the named classes (the ones with rdf:ID) are transformed in relations. Furthermore, generalization can be mapped to the relational model in many ways and all the choices are available to the user.

We choose the following translation process. The description we provide here refers to the constructs of the source and the target models, to clarify how the translation is performed but we remark that the actual translation steps are defined on the meta-representation. Some samples of Datalog rules are illustrated at the end of this section. For each named class C a relation C' is created with an assigned OID as primary key. The attribute `OID` is

generated automatically and has the role of filling the absence of primary keys in the OWL-Lite model. The single value datatype properties a are mapped to the columns a', while multivalued properties became a separated relation A' with a reference column to the relation C'. If the class C is subclass of D, the relations D' and C' are created but C' has not an `OID` column assigned because it is externally identified by the relation D'. If there are two classes C and D related by an object property b. The classes C and D are translated to the relations C' and D'. If the object property b is functional from C to D, it is mapped into a foreign key column b' assigned to C', that refers to the OID of the relation D' and is `NOT NULL`. If b is also inverseFunctional, then b' is defined `UNIQUE, NOT NULL`. If the property that relates two classes, C and D, is a simple `owl:ObjectProperty`, it can be seen as a *many to many* relationship, then will be mapped in a separate relation B' that has two attributes c', d', with foreign keys to the OIDs of the relations C' and D'.

The above process is executed as a Datalog program that is composed by simple rules that represent each step.

The two examples of simple rules we provide here, refer to the cases of translating named classes:

SM_Aggregation(
OID:#AggregationOID_1*(oid),
Name: name
)
←
SM_Abstract(
OID: oid,
Name: name
),
SM_Lexical(
OID: lexicalOID,
Name: "rdf:ID",
isIdentifier: "true",
abstractOID: oid
)

and a functional `owl:ObjectProperty`:

```
SM_ComponentOfForeignKey(
        OID: #ComponentOfForeignKeyOID_1
                *(oid, lexOID),
        Name: name,
        foreignKeyOID: #ForeignKeyOID_1(oid),
        lexicalFromOID: #LexicalOID_1(oid,lexOID),
        lexicalToOID: lexOID
)
←
SM_BinaryAggregationOfAbstracts(
        OID: oid,
        isFunctional1: "true",
        abstract2OID: abs2OID
),
SM_Lexical [DEST] (
        OID: lexOID,
        Name: name,
        isIdentifier: "true",
        type: t,
        aggregationOID: #AggregationOID_1(abs2OID)
),
SM_Abstract(
        OID: abs2OID
)
```

The meaning of the Datalog rules should be quite intuitive for the reader but let us explain some details. The terms that are preceded by the # character, are the skolem functors that have the role of generating an invented identifiers, that is always the same when passing the same argument. The first rule selects all the `SM_Abstract` that have a `SM_Lexical` with the name `rdf:ID` and generates a correspondent `SM_Aggregation`. The second rule generates a `SM_ComponentOfForeignKey`

The execution of the rules generates the target schema but still expressed in terms of meta-constructs. The last step is a simple transformation that copies this meta-representation into a one that is expressed in terms of the target model.

Supermodel To Relational

Exploiting the correspondences between the supermodel and the relational model (that are sketched in figure 5.4), it is possible to generate the rela-

tional schema that is illustrated in the following:

Person(OID, Name, Surname)
SoccerPlayer(PersonOID, Score, SoccerTeamOID)
SoccerTeam(OID, TeamName, SoccerPlayerOID)
PlayedInThePast(SoccerPlayerOID, SoccerTeamOID)

with the following constraints:

SoccerPlayer.SoccerTeamOID *references* SoccerTeam.OID
SoccerTeam.SoccerPlayerOID *references* SoccerTeam.OID
PlayedInThePast.SoccerPlayerOID *references* SoccerPlayer.OID
PlayedInThePast.SoccerTeamOID *references* SoccerTeam.OID

As we previously underlined, the obtained schema is only an example of relational schema that can be generated, choosing a different set of rules will lead to a different result, following the user needs.

5.5 From Relational Database to OWL Ontology

As we briefly said in the introduction, an important issue about interoperability between Semantic Web and databases regards the exportation of information stored in databases to a form that is interpretable from Semantic Web applications.

In this section we aim at explaining how our approach is suitable to perform translations between relational databases and OWL ontologies. As we stated in the previous section the approach is not dependent from the specific model as the translations are made within the supermodel.

The three steps of the translation process in this case are: i) importing relational constructs in the supermodel, ii) translation between supermodel constructs, iii) exporting from the supermodel to OWL.

Let us consider the following relational example:

Professor(ID, Name, Surname)
Course(CourseID, CourseName, ProfID)
Project(IDProj, ProjName)
Participate(ProfessorID, ProjectID)

with the following constraints:

Course.ProfID *references* Professor.ID
Participate.ProfessorID *references* Professor.ID
Participate.ProjectID *references* Project.IDProj

The example describes the relationship between Professors and their Courses and Projects.

Relational to Supermodel

In this step we want to describe the relational constructs in terms of the supermodel constructs. Due to the fact that all modifications we have applied to the supermodel only augment its capabilities, keeping a backward compatibility, we don't need to re-write the correspondences between the relational model and the supermodel but just reuse the previously defined ones. Therefore for the initial step of "copying" information from the relational model to the supermodel we can take advantage of the previous work (for more details see [**?**]).

The correspondences between the relational model and the supermodel are sketched in Fig. 5.4

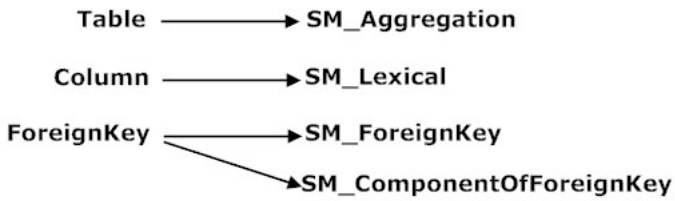


Figure 5.4: Correspondences between the relational model and the supermodel.

Translation within the Supermodel

The second step is the real translation that must be made in term of supermodel constructs.

As the relational model was already considered in the previous work, the translation rules were partially reused, some were updated and only few rules have been written from scratch.

At a simple level, we can state that each relational table can be translated in an OWL class. Therefore, for each relation R in the relational model, we must have a class C in the OWL model. The attributes of the relations (the columns) a_1, that are not foreign key, are mapped to datatype property a_1' in OWL. Each attribute a_2 of a relation R_1, that is foreign key of a relation R_2, is mapped as an object property a_2' that relates the two classes C_1 and C_2, derived respectively from relations R_1 and R_2.

As the concept of primary key is not present in OWL-Lite, we loose this construct during the translation. Actually the only case where it is possible to keep the concept of relational primary key in an OWL ontology is when the primary key is composed by a single attribute. Let a_k be the primary key of a relation R. The naif method to translate it to OWL-Lite is to define a class to map the relation C_R, another class to map the key attribute C_k. Then a functional object property O from C_R to C_k and its inverse functional O', implement the meaning of the primary key. Since it is not possible to generalize this approach to cases when the key is composed by more attributes, we choose not to keep the primary keys in OWL-Lite.

```
<owl:ObjectProperty rdf:ID="participate">
    <rdfs:domain rdf:resource="#Professor"/>
    <rdfs:range rdf:resource="#Project"/>
    <owl:inverseOf rdf:resource="#inverseOf_participate"/>
</owl:ObjectProperty>

<owl:ObjectProperty rdf:ID="inverseOf_participate">
    <rdfs:domain rdf:resource="#Project"/>
    <rdfs:range rdf:resource="#Professor"/>
</owl:ObjectProperty>
```

All steps we have described (and some other that we omit for the lack of space) are realized through a Datalog program, that is composed of simple rules that execute each elementary step. In order to give an idea of the rules for this kind of translation we present two examples.

The first rule is used to translate a relation into an OWL class. At the supermodel level we have that each relation can be mapped as Aggregation and each OWL class to Abstracts. Therefore we need to define a rule to map the `SM_Aggregation` construct to `SM_Abstract` construct as follows:

SM_Abstract(
OID:#AbstractOID_3*(oid),
Name: aggregationName
)
←
SM_Lexical(
OID: lexicalOID,
Name: lexicalName,
isNullable: isNullable,
isIdentifier: "false",
aggregationOID: aggregationOID
),
!SM_ComponentOfForeignKey(
lexicalFromOID: lexicalOID
),
SM_Aggregation(
OID: oid,
Name: aggregationName
)

The second rule is used to translate the attributes of a relation that are not foreign keys of other relations.
Therefore each `Lexical` of an `Aggregation`, which is not a `ComponentOfForeignKey`, is translated into a `Lexical` (in other words an attribute of a relation which is not a foreign key in relational model can be translated into an `owl:DatatypeProperty`. The rule is defined as follows:

```
SM_Lexical(
        OID:#LexicalOID_0*(lexicalOID),
        Name: lexicalName,
        isOptional: isNullable,
        isIdentifier: isId,
        type: type,
        abstractOID:#AbstractOID_3(aggregationOID)
)
←
SM_Lexical(
        OID: lexicalOID,
        Name: lexicalName,
        isNullable: isNullable,
        isIdentifier: isId,
        aggregationOID: aggregationOID
),
!SM_ComponentOfForeignKey(
        lexicalFromOID: lexicalOID
),
SM_Aggregation(
        OID: aggregationOID)
)
```

Supermodel to OWL

This step is quite similar to the one described in section 5.4. Indeed the last step of the translation process is a copy operation that produces the target schema using the constructs of the target model. The final result is an OWL-compliant ontology that maintains the semantics of the source relational database.

5.6 Conclusions

In this thesis we showed how MIDST, an implementation of the ModelGen operator that supports model-generic translations of schemas and their instances within a large family of models, can be extended to address the large family of Semantic Web formalism. In particular we show how our approach can be suitable to perform translation from relational databases to OWL ontologies and viceversa. Following the research illustrated in this chapter, we published the article "Ontologies And Databases: Going Back And Forth" (see [**?**] for details).

Chapter 6

Temporal aspects for data intensive Web sites

The adoption of high-level models, for temporal, data-intensive Web sites is proposed together with a methodology for the design and development through a content management system (CMS). The process starts with a traditional E-R scheme; the various steps lead to a temporal E-R scheme, to a navigation scheme and finally to a T-ADM scheme. The logical model allows the definition of page-schemes with temporal aspects (which could be related to the page as a whole or to individual components of it). Each model considers the temporal features that are relevant at the respective level. A content management tool associated with the methodology has been developed: from a typical content management interface it automatically generates both the relational database (with the temporal features needed) supporting the site and the actual Web pages, which can be dynamic (JSP) or static (plain HTML or XML), or a combination thereof. The tool also includes other typical features of content management all integrated with temporal feature.

6.1 Introduction

The systematic development of Web sites has attracted the interest of the database community as soon as it was realized that the Web could be used as a suitable means for the publication of useful information of interest for community of users (Atzeni et al. [**?**], Ceri et al. [**?**], Fernández et al. [**?**], Brambilla et al. [**?**]). Specific attention has been devoted to *data-intensive*

sites, where the information of interest has both a somehow regular structure and a possibly significant volume; here the information can be profitably stored as data in a database and the sites can be generated (statically or dynamically) by means of suitable expressions (i.e. queries) over them (Merialdo et al. [**?**]). In this setting, the usefulness of high-level models for the intensional description of Web sites has been advocated by various authors, including Atzeni et al. [**?**, **?**] and Ceri et al. [**?**, **?**], which both propose logical models in a sort of traditional database sense and a *model-based development* for data intensive Web sites.

When accessing a Web site, users would often get significant benefit from the availability of time-related information, in various forms: from the history of data in pages to the date of the last update of a page (or the date the content of a page was last validated), from the access to previous versions of a page to the navigation over a site at a specific past date (with links coherent with respect to this date). As common experience tells, various aspects of a Web site often change over time: (i) the actual content of data (for example, in a University Web site, the instructor for a course); (ii) the types of data offered (at some point we could decide to publish not only the instructor, but also the teaching assistants, TAs, for a course); (iii) the hypertext structure (we could have the instructors in a list for all courses and the TAs only in separate detail pages, and then change, in order to have also the TAs in the summary page); (iv) the presentation. Indeed, most current sites do handle very little time-related information, with past versions not available and histories difficult to reconstruct, even when there is past data. Clearly, these issues correspond to cases that occur often, with similar needs, and that could be properly handled by specific techniques for the support to time-related aspects. We believe that a general approach to this problem, could generate a significant benefit to many data-intensive Web applications.

Indeed, we have here requirements that are analogous to those that led to the development of techniques for the effective support to the management of time in databases by means of *temporal database* (see [**?**], [**?**], [**?**] and [**?**] for interesting discussions).

It is well known that in temporal databases there are various dimensions along which time can be considered. Beside *user-defined time* (the semantics of which is "known only to the user", and therefore is not explicitly handled), there are *valid time* ("the time a fact was true in reality") and *transaction time* ("the time the fact was stored in the database").

In order to highlight the specific aspects of interest for Web sites, we concentrate on valid time, which would suffice to show the main ideas. Transaction time would have similar requirements, plus some specific, additional facets.

In a Web site, the motivation for valid time is similar to the one in temporal databases: we are often interested in describing not only snapshots of the world, but also histories about its facts. However, there is a difference: in temporal databases the interest is in storing histories and in being able to answer queries about both snapshots and histories, whereas in Web sites the challenge is on how histories are offered to site visitors, who browse and do not query. Therefore, this is a design issue, to be dealt with by referring to the requirements we have for the site. The natural (and not expensive) redundancy common in Web sites could even suggest to have a coexistence of snapshots and histories.

Temporal Web sites require the management of temporal data and especially its collection. In this respect, it is worth noting that most Web sites are supported by applications that handle their data (and the updates to them). These applications are often implemented with the use of a Content Management System (CMS). We claim that the extension of CMSs with the explicit management (acquisition and maintenance) of time-related data can provide a significant contribution to our goal. This would obviously require the representation, in the CMS repository, of the temporal aspects of the information to be published.

This work aims at giving a contribution to the claim that the management of time in Web sites can be effectively supported by leveraging on the experiences made in the database field, and precisely by the combination of the three areas we have briefly mentioned: temporal databases, content management systems and model-based development of Web sites. In particular, attention is devoted to models and design: models in order to have a means to describe temporal features and design methods to support the developer in his/her decisions on which are the temporal features of interest to the Web site user. The approach relies on a CASE tool that handles the various representations and transformations and so it gives significant support to the designer in the development of the various components.

This section of the thesis extends the experiences in the Araneus project [**?**, **?**, **?**, **?**] where models, methods and a CMS prototype for the development of data-intensive Web sites were developed. Indeed, we propose a logical model for temporal Web sites, a design methodology for them and a tool to support the process. A first version of the tool has been recently demonstrated (Atzeni and Del Nostro [**?**]). We refer here to a new version that also provides a CMS-style interface to support the user's modifications to the site content and gets temporal information from these actions.

The rest of the chapter is organized as follows. Section 6.2 is devoted

to a brief review of the aspects of the Araneus approach that are needed as a background. Then, Section 6.3 illustrates the temporal extensions for the models we use in our process and Section 6.4 the specific CMS features. In Section 6.5 the methodology with the associated tool is illustrated by means of an example of usage. Finally, in Section 6.6 we briefly sketch possible future developments.

6.2 The Araneus models and methodology

The Araneus approach (Merialdo et al. [**?**]) focuses on data-intensive Web sites and proposes a design process (with an associated tool) that leads to a completely automatic generation of the site extracting data from a database. The design process is composed of several steps each of which identifies a specific aspect in the design of a Web site. Models are used to represent the intensional features of the sites from various points of view:

1. the Entity Relationship (ER) model is used to describe the data of interest at the conceptual level (then, a translation to a logical model can be performed in a standard way, and is indeed handled in a transparent way by the associated tool);

2. a "navigational" variant of the ER model (initially called NCM and then N-ER) is used to describe a conceptual scheme for the site. The main constructs in this model are the major nodes, called *macroentities*, representing significant units of information, which often consolidate concepts from the ER model (one or more entities/relationships), and navigation paths, expressed as *directed relationships*. Nodes of an additional type, called *aggregations*, are used to describe the hierarchical access structure of the hypertext;

3. a logical scheme for the site is defined using the Araneus Data Model (ADM), in terms of *page schemes*, which represent common features of pages of the same "type" with possibly nested attributes, whose values can come from usual domains (text, numbers, images) or be links to other pages.

The design methodology (sketched in Figure 6.1, see Atzeni et al. [**?**]), supported by a tool called Homer (Merialdo et al. [**?**]), starts with conceptual data design, which results in the definition of an ER scheme, and then proceeds

with the specification of the navigation features, macroentities and directed relationships (that is, a N-ER scheme). The third step is the description of the actual structure of pages (and links) in terms of our logical model, ADM.

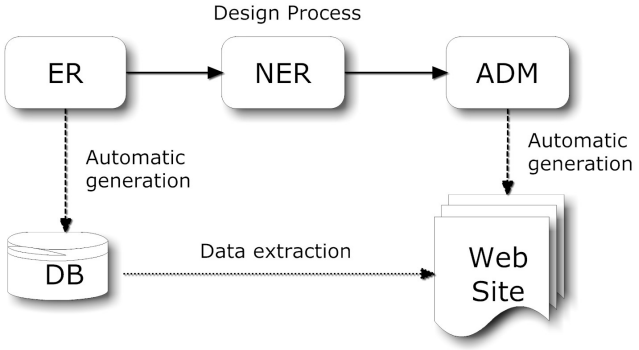


Figure 6.1: The Araneus design process

Three simple schemes for the Web site of a University department, to be used in the sequel for comments, are shown in Figures 6.2, 6.3, and 6.4, respectively.

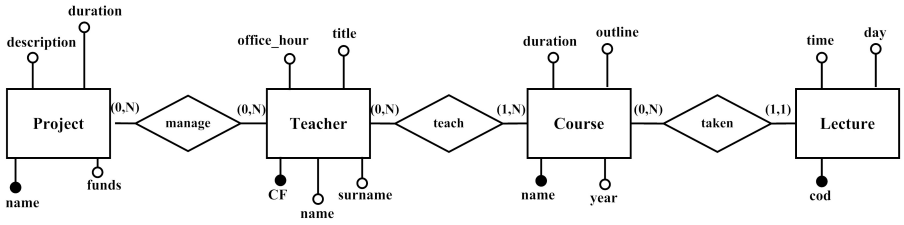


Figure 6.2: The example of ER schema

A fourth step is the specification of the presentation aspects, which are not relevant here. In the end, since all the descriptions are handled by the tool and the various steps from one model to the other can be seen as algebraic transformations, the tool is able to generate, in an automatic way, the actual code for pages, for example in JSP or in plain HTML, with access to a relational database built in a natural way from the ER scheme.

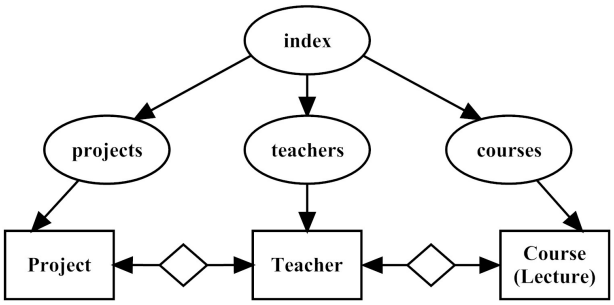


Figure 6.3: The example of N-ER schema

6.3 Models for the management of temporal aspects of Web sites

Temporal aspects appear in all phases of the design process and therefore each of our models needs a suitable extension. We consider the point of views of both types of users that are involved in a Web site content evolution: final users, who access the site and would have tools to explore the content (and its changes), and content administrator, who apply content modifications. In our approach the designer can specify how temporality should be managed and presented to the final user and, at the end of the process, a temporal web site is automatically produced together with the related content management application that can be used by content administrator. Our goal is to have a "standard" Web site with features that simplify the management of temporal aspects. Considering that the most common use of the Web site will involve current values, the introduction of temporal aspects should not negatively impact the site structure complexity. We start with brief comments on the models we use for describing our data and then illustrate the conceptual and logical hypertext models that have been properly extended to allow the representation of temporal aspects.

Models for the representation of data

Different kinds of representations have been proposed to manage temporal aspects at a conceptual level, each of which with its specific features. Some of the models (see Gregersen and Jensen [**?**] for a survey) represent a temporal object in the schema and allow users to define temporal elements by rather

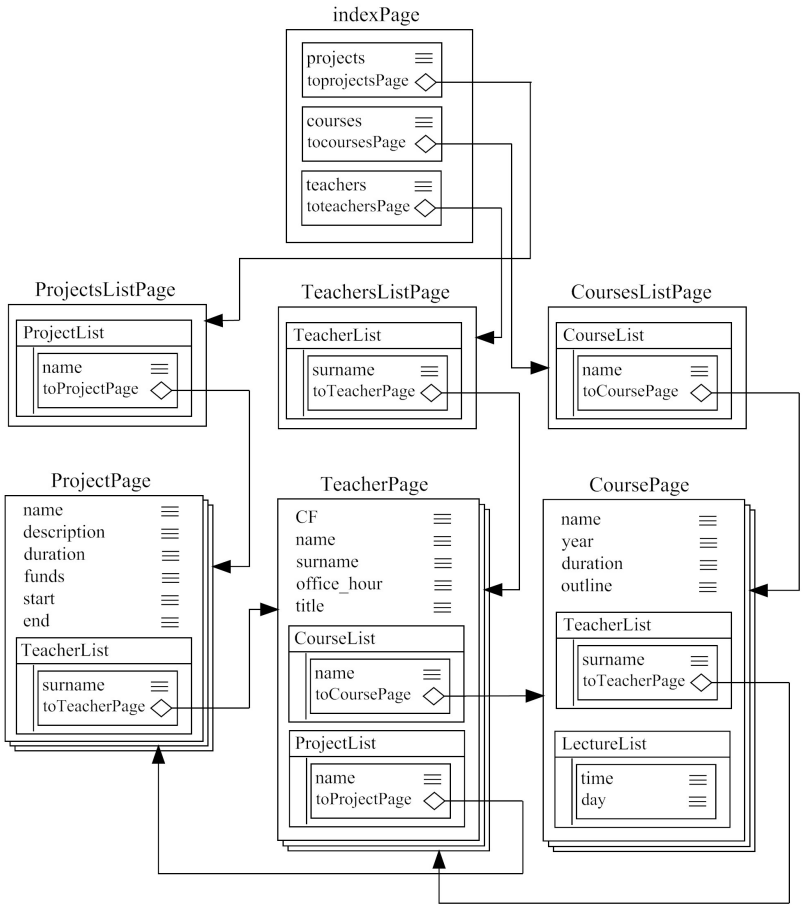


Figure 6.4: The example of ADM schema

complex visual notation that is not suitable for users with no specific competence or that need a steep learning curve. Other approaches are based only on textual notation that is used to specify temporal properties beside the snapshot conceptual modeling. While the schema readability is not compromised, users need to jump between the two models to know which concept has temporal features.

As is illustrated by Artale et al. [**?**], there is a gap between the modeling solutions provided by researchers and the real needs of the users. A conceptual model should be simple and expressive, in order to allow users to represent concepts, properties and relations by a visual interface with intuitive constructs. The diffusion of the ER model is based right on this characteristics. Therefore, in the conceptual design phase, we choose an extension of the ER model where temporal features are added to the scheme, by indicating which are the entities, relationships and attributes for which the temporal evolution is of interest. With our conceptual model proposal we don't aim at covering all the aspects that can arise when facing with temporal evolution of content but providing users with some intuitive tools that allow the management of temporal aspects within a site design process. For each object in a scheme, the model allows to specify whether it is temporal. To keep the conceptual model as simple as possible we separate the specification of temporal elements from the definition of temporal properties for which we provide a textual notation. We briefly illustrate the main temporal constructs we have considered in our framework. A temporal object 0^T can be defined either as a single attribute, entity, relationship. In the schema, temporal objects are identified by an uppercase T as a superscript. To allow users to define a time granularity for an object, we provide the chronos construct with the syntax `CHRONOS`(G, 0^T) where G is the time granularity and 0^T is the temporal object. Examples of time granularity G are "day", "week", "month", "year". For example, if an object is defined as temporal and the month chronos is specified, then one value per month will be considered for publishing. So looking at the so defined temporal schema, it can be immediately noticed which are the objects for which the evolution is of interest, keeping the schema simplicity without compromising readability. A separated textual notation is devoted to the refining of temporal properties, as is in the standard ER model where concepts are illustrated graphically while constraints are specified in a textual way. In Figure 6.5 an example usage of the notation is sketched.

Let's start a quick explanation of what the schema express. The designer wants to keep the *Project.duration* evolution that can be changed, for example due to an extension of the ending time. Notice that in this model it is possible to specify temporal elements at different levels of granularity either for single attributes or for an entity (or a relationship) as a whole. Moreover, as illustrated in Figure 6.5, an entity and its attributes can be independently defined as temporal: this is the case for Teacher and its attributes. Temporality of the entity means that we are interested in its life cycle: in the example, in the instants when an individual became a teacher in our school and when he

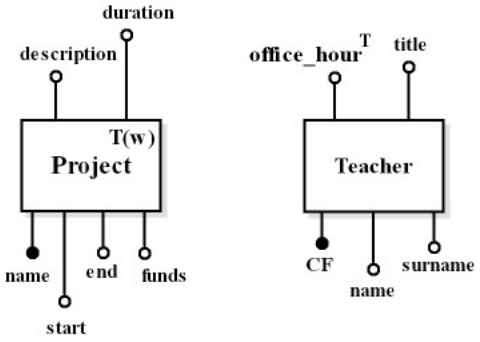


Figure 6.5: Temporal notation example

quits or retires (but we will keep information after that). If an entity is not temporal, we are not interested in its evolution; should it be deleted, it would be completely discharged. For the attributes, the interest in the evolution of values (the address, in the example).

If a single attribute is marked as temporal, its evolution is maintained with independently of other attributes. For example if every year the project manager can be changed then the designer specify that the `CHRONOS` for the Manage relation is Y (year).

The database used to handle the data for our Web sites is relational, as in the Araneus approach, with temporal features added to it (if using our tool the temporal features are generated automatically and the developer need not have any specific competence). Optimization aspects for temporal databases are beyond our focus, thus a simple relational schema for temporal tables has been chosen. When an entity is specified as temporal, then two timestamps are added to the relational representation, in order to define the beginning and the end of the validity interval and the original primary key is extended including those timestamps. For each temporal attribute an additional table is created to manage its modifications separately.

Models for the representation of Web sites

We use two models to describe the structure of a Web site at different levels of abstraction: the N-ER model considers concepts whereas ADM refers to the actual structure of pages. The same distinction applies to their temporal

extensions and (as they are tightly related) to the specific CMS aspects that concern modifications management. Therefore the conceptual representation of the Web site (N-ER model) is extended to allow the selection of the versions of interest for a concept. Then, in the logical model (ADM), new constructs are introduced to give the designer the possibility to choose how to organize versions in pages.

Temporal aspects of Web sites at a conceptual level

The temporal N-ER model allows the specification of whether versions have to be managed for the concepts (macroentities and directed relationships) of interest for the site, and how.

There are two main versioning aspects involved in this model. The first refers to the possibility that an object can be modified and the second concerns the inclusion of the evolution of changes in the site. These two points of view do not necessarily coincide. Based on the choices expressed in the T-ER model, different possibilities exist. If an object is identified as snapshot, it is here possible to choose whether it can change or not. Specifying here that a snapshot object is modifiable means that updates are allowed but the temporal database will not store the history of changes. In our example we could have that the *description* attribute of the *Project* entity is defined as snapshot in the T-ER model, because there is no interest in keeping track of its versions, but we want the system to allow changes and the Web user to be informed when the last changes happened (logical level aspects will be detailed in the next section). This is an example of whether a snapshot object should be considered as *modifiable* in the T-NER model.

In this model it is possible to define the temporality features for each of the temporal concepts (macroentities, direct relationships, and attributes). A concept can be defined as temporal if its origin in the T-ER scheme is a temporal component, but not necessarily vice versa. Therefore, we could have macroentities that are not defined as temporal even if they involve temporal elements, for example because the temporality is not relevant within the macroentity itself (indeed, Web sites often have redundancy, so an attribute or an entity of the ER scheme could contribute to various macroentities, and, even if temporal, it need not be temporal in all those macroentities). Consider the case where the designer needs to keep the history of all the modifications to the project information then the *Project* entity is defined as temporal in the T-ER model. The project data are used in two macroentities, *ProjectsListPage* and *ProjectPage*. The *ProjectsListPage* macroentity is just a list of projects with

the project name as a link label, and therefore will not have any version management. The *ProjectPage* macroentity gives the user all the details about a project and it is here interesting to have a management of versions in order to let the user know about the changes.

It is worth noting that these are essentially "patterns", which correspond to solutions, and others can be added if needed.

For macroentities and attributes, a major facet is relevant here: which versions are of interest from the Web site conceptual point of view? We consider this as a choice from a set of alternatives, which currently include (i) none; (ii) the last version; (iii) all versions.

Till now we have considered temporal aspects of each concept separately (how to manage versions for each temporal element) but this need not be the only approach. An alternative would be navigation with respect to a specific instant, neglecting the others. Let us explain the idea by an example. Consider the following situation of the university Web site where both the Teacher page and the Course page are temporal: the teacher Smith page has two versions each of which with its validity interval as in Figure 6.6, the Database course given by Smith has the two versions illustrated in Figure 6.7 (dashed boxes highlight the changes).

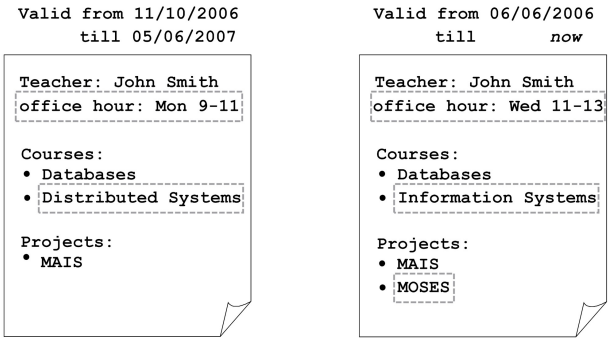


Figure 6.6: Versions for the TeacherPage

If visiting the site we are on the Smith page we can select to view only information valid in a specific day. Then, we might want to keep on navigating with reference to that same day. Following the link to the Database course page, only data valid in the previously selected instant should be published. Navigating the site at the current time the navigation path is shown in the

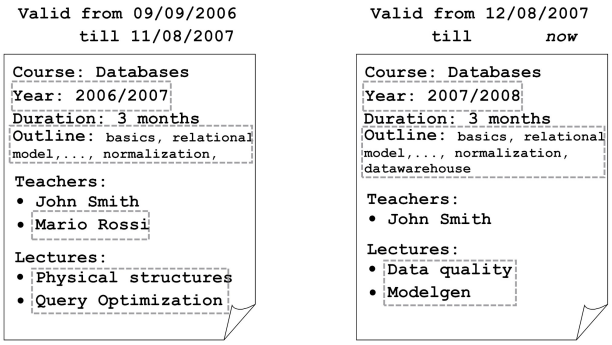


Figure 6.7: Versions for the CoursePage

upper part of Figure 6.8. Choosing the validity instant 04/04/2007 the correct navigation path is sketched in the lower part of Figure 6.8. This kind of navigation is called "time-based selective navigation" between pages.

Temporal aspects of Web sites at a logical level

The logical design of a temporal Web site has the goal of refining the description specified by a temporal N-ER scheme, by introducing all the details needed at the page level: how concepts are organized in pages and how versions of temporal elements are actually published. The temporal extension of ADM (hereinafter *T-ADM*) includes all the features of ADM (and so allows for the specification of the actual organization of attributes in pages and the links between them), and those of the higher level models (the possibility of distinguishing between temporal and non-temporal page schemes, and for each page, the distinction between temporal and non-temporal attributes; since the model is nested, this distinction is allowed at various levels in nesting, apart from some technical limitations), and some additional details, on which we concentrate. A major choice here is the implementation of the versioning requirement specified at the conceptual level.

As this model can be obtained as a translation from the higher level models, the temporal choices expressed on the previous models drive the various alternatives offered:

(a) No versioning

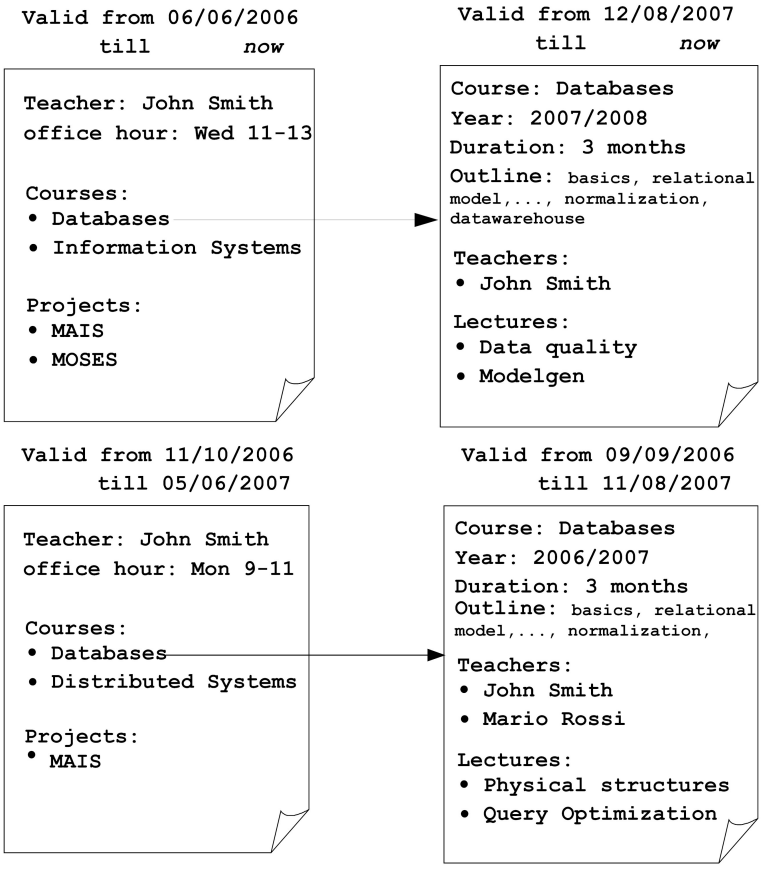


Figure 6.8: Two examples of temporal navigation

(b) Last version

(c) All versions in the same page

(d) All versions in separate pages

(e) Time based selective navigation

In case (a) no version management is required; this is actually a consequence of the decision of not considering versions in the temporal N-ER model. When only the last version is of interest, then the designer expresses choice (b). This can regard either temporal elements, when the designer does not want to publish previous versions but only the current value, or snapshot elements in the case the modification information are needed in the page. When the N-ER design choice for a temporal element is to manage all versions, the alternative (c) allows the possibility to include them together in the same T-ADM page scheme. This means that the designer gives all the versions the same importance thus the user will catch them in a single view. If the current and the previous versions have different browsing priority, it is possible to separate the "current value" from the previous, correlated by means of links, choosing alternative (d). Various browsing structures, which will be detailed later on, allow different ways to publish versions according to the designer choices. A completely different organization is the "time-based selective navigation", represented by option (e). In this case the user selects, for a page, the instant of interest and sees the corresponding valid versions. The navigation between temporal pages can then proceed keeping reference to that instant.

Additional features allow for the emphasization of recent changes (on a page or on pages reachable via a link). The above features are expressed in T-ADM by means of a set of constructs, which we now briefly illustrate.

We first illustrate the T-ADM extensions that allow the designer to choose which meta data have to be published with versions: CREATOR, MODIFIER, DESCRIPTION, as we briefly illustrate in the following:

CREATOR Represents the creator of a content. This information can just be associated with the current value of an information or (introducing a bit of redundancy) with each of its versions.

MODIFIER This attribute is related to a version and represents the author of the change to a data element.

DESCRIPTION It can be related to an element that can be changed (temporal or snapshot with the differences illustrated in the previous section) to publish the reason of the modification.

We also have two constructs representing two different temporal pieces of information that can be associated with the content version:

LAST MODIFIED This is a special, predefined attribute used to represent the date/time (at the granularity of interest) of the last change applied to a

temporal element. This is a rather obvious, and widely used technique, but here we want to have it as a first class construct offered by the model (and managed automatically by the support tool) and also we think it should be left to the site designer to decide which are the pages and/or attributes it should be actually used for, in order to be properly informative but to avoid overloading.

VALIDITY INTERVAL This is another standard attribute that can be associated with any temporal element.

The next is a major feature of the model, as it is the basis for the time-based selective navigation:

TIME POINT SELECTOR It can be associated with pages and with links within them, in such a way that navigation can proceed with reference to the same time instant; essentially, in this way the user is offered the site with the information valid at the specified instant.

Another feature is used to highlight a link when the destination is a page that includes temporal information which has recently (according to a suitable metric: one day, one week, or whatever the designer chooses) changed:

TARGET CHANGED This property can be used in association with LAST MODIFIED to add the time the modification has been applied.

The TARGET CHANGED feature is illustrated in Figure 6.9: a Department page (source) has a list of links to teacher pages (target). In a teacher page the office hours have been modified. When the user visits the department page he is informed which teacher pages have been modified (and when) so he can follow the link to check what is new. The example refers to just one source and one target page, but things may become more interesting when we consider non-trivial hypertextual structures: this gives the opportunity to propagate this kind of information through a path that leads to the modified data (see Figure 6.10). When a new lecture is introduced, then both the teacher and the department page are informed (and highlight the change) so the user can easily know which are the site portions with modified data.

We now illustrate the additions brought to the model to implement the different ways versions can be presented to the user:

REVISION LIST This feature allows for the specification that all versions of a temporal element are shown in the same page as a list of revisions.

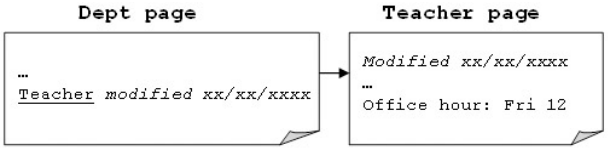


Figure 6.9: The TARGET CHANGED feature

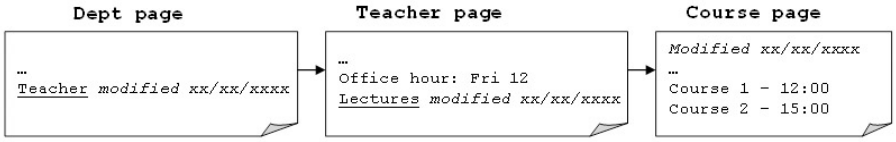


Figure 6.10: The TARGET CHANGED feature along a path

LINK TO VERSIONS This is a special type of link that has as a target a VERSION STRUCTURE (to be illustrated shortly), handling the versions of a temporal element. It is used when the designer chooses to have just the last version in the main page and the others held in other pages.

VERSION STRUCTURE These are "patterns" for pages and page schemes, used to organize the different versions of a temporal element and referred to by the LINK TO VERSIONS attribute. There are various forms for this construct involving one or more pages:

- SIMPLE VERSION STRUCTURE: a single page presenting all the versions for the temporal element with timestamps.
- LIST VERSION STRUCTURE: an "index" page with a list of links labeled with the validity intervals that point to pages showing the particular versions and include links back to the index.
- CHAIN VERSION STRUCTURE: this is a list of pages each of which refers to a specific version. It is possible to scan versions in chronological order, by means of the "previous" and "next" links available in each page.
- SUMMARY VERSION STRUCTURE: similar to the previous case but the navigation between versions is not chronological. Each version page has a list of links that works as an index to all versions.

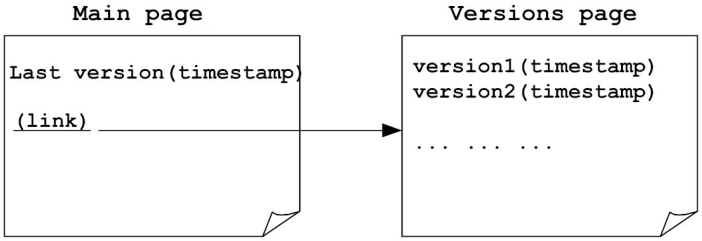


Figure 6.11: The SIMPLE VERSION STRUCTURE pattern

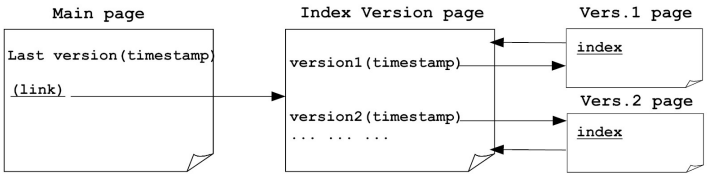


Figure 6.12: The LIST VERSION STRUCTURE pattern

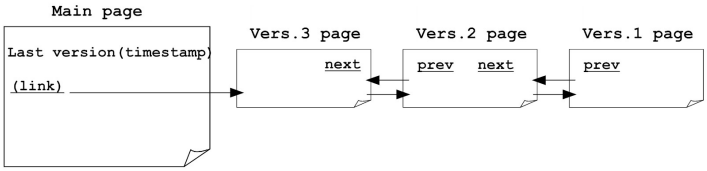


Figure 6.13: The CHAIN VERSION STRUCTURE pattern

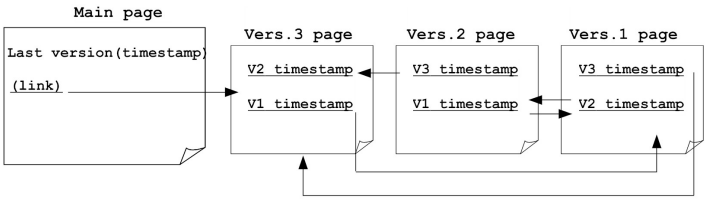


Figure 6.14: The SUMMARY VERSION STRUCTURE pattern

As we mentioned earlier, the designer is guided during the design process. The tree of available choices, along the T-Araneus process, that can be expressed at each abstraction level is sketched in Figure 6.15.

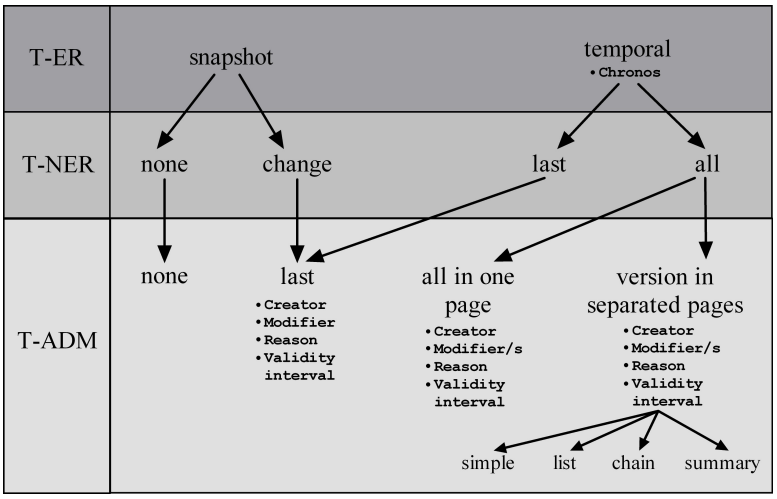


Figure 6.15: The tree of available choices

6.4 CMS support to T-Araneus

The need for Content Management Systems (CMS) as fundamental tools for handling the information to be published in Web sites was recognized soon after the establishment of the Web as a strong communication medium [**?**, **?**, **?**]. In data-intensive Web sites this would mean that the updates to the database should be controlled in a rigorous way. From our point of view, it is worth noting some features usually available in CMSs and a special requirement often neglected by them.

One of the goals of a CMS is to support users in the changes to the content, allowing the management of the whole life cycle of these contents also through workflow management capabilities.

The most relevant feature of these systems, which is very important for our purposes, is that they can handle the history of a site and of its contents (and the associated responsibilities: "who changed a piece of content and when?").

However CMSs usually have a lack of flexibility in handling content granularity. In particular they consider pages or documents as the granularity of interest. Instead, we believe that, in many cases, especially for data-intensive Web sites, different levels of granularity are needed: individual pieces of data, corresponding to atomic fields in databases, a table in the related database, the whole page or even the entire site.

Starting from the Araneus tool, whose approach is discussed in Section 6.2, used to generate a site from the definition of the aforementioned models, we have developed a new CASE tool that also handles the temporal information of a site. Moreover, to take profit from the CMS features this tool now generates both the site and the temporal CMS that is required to manage the site itself. In other words, an instance of our CMS is generated along with the Web site.

In particular we extend the functionality related to the management of temporal features that can be associated to content at different level of granularity.

Temporal Content Management

In our work, we have extended the typical features of CMSs focusing on two aspects: the management of data-intensive sites and the management of temporal information at different level of granularity, from the finest (single piece of data in the underlying database) to the coarsest (potentially the entire site).

Our CMS guarantees the needed support for handling histories of data at the needed granularity. Moreover, with a fine granularity, also the structure of pages in the CMS could become complex and delicate to design, but it can be supported by the models and methodology we have discussed earlier.

The high level models we explained in the previous sections are fully supported through the CMS features. Given the fine granularity of data and the possible redundancy in the data in the site, non-trivial design choices arise for the definition of the pages of the CMS: which pages are needed (for example, the instructor for a course appears in various pages; do we want to have CMS pages for updating it corresponding to each of the Web site pages or just one of them suffices?), with what attributes, and so on. At the same time, by incorporating the CMS in our methodology, we can derive some decision on structures by considering the temporal properties of pieces of data: if an attribute cannot change, then there is no need for a form to handle the updates.

The use of our CMS can greatly improve the integration of time into the World Wide Web. This should enable the user to reconstruct historical content and follow its evolution.

From the CASE tool interface it is possible to design the entire site, there-

fore the designer can define all temporal features from the different scheme views (from ER to N-ER and finally to ADM). Therefore the site is automatically generated from the data stored in the underlying database and the necessary tables for storing of the temporal features are also created. Then the CMS pages are generated and from this point all contents of the site can be managed through the CMS functionalities.

As the CMS is automatically generated on the basis of the designer's choices at the time he/she specifies the temporal features of the site, these choices also modify both the presentation forms (all versions in a page, versions in separated linked pages) and the update/deletion forms of the content.

An important observation is that, by managing all updates, our CMS can easily produce and maintain the meta-data of interest for the generation of the pages of our sites, some of which have been mentioned in the previous sections: the author and modifier of a piece of data, the time information for a given change, the motivation for it.

Moreover, our temporal CMS supports some typical and useful features of a standard CMS, namely: workflow management, user administration.

One of the most typical features of a CMS is the possibility to access the content at various levels and with different rights, through a user management systems. Our system is based on the definition of different classes of users, that can be managed through the CMS interface. In particular there are three main classes: (i) Administrator: the users of this class have access to all managing features and to all contents without restriction. They can also define new users or users classes and define the respective authorizations. (ii) Editor: these users will be the managers of individual sites, and can be considered as administrators of their sites; (iii) Author: they are users that can create or modify only the contents of a specific area of a site (assigned by an Editor). Each change creates a temporal version of a content according to the chosen aforementioned methods. Our CMS has also a Workflow System that enables the collaborative management of contents and temporal information. A workflow instance can be represented through the model shown in Figure 6.16. An *activity* represents the needed operation to complete the workflow task; namely we distinguish between (i) *atomic activities* i.e. a set of operations required by the workflow, that can be combined so that they appear to the rest of the system to be a single operation, (ii) *complex activities* that imply the existence of nested workflows. Unlike traditional workflow systems that are available to a CMS, our workflow activities may affect the contents manipulation at different levels of granularity, from the coarsest (the single Web page) to the finest (the single record in the database).

Transition conditions define the modalities of transformation from one state to another of an activity. The possible transition are (i) *Inactive*: the activity has been created but not activated; (ii) *active*: the activity has been created and activated; (iii) *suspended*: the activity is suspended and can not complete the operations.

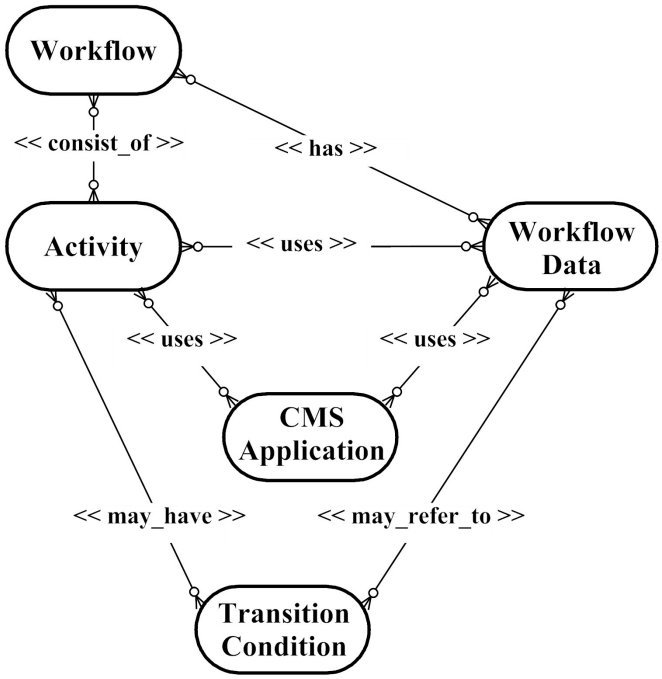


Figure 6.16: Workflow model

Architecture of the system

In Figure 6.17 a sketch of the architecture of our implemented prototype is presented.

The architecture is characterized by three different layers, namely: i) repository, ii) application and iii) user.

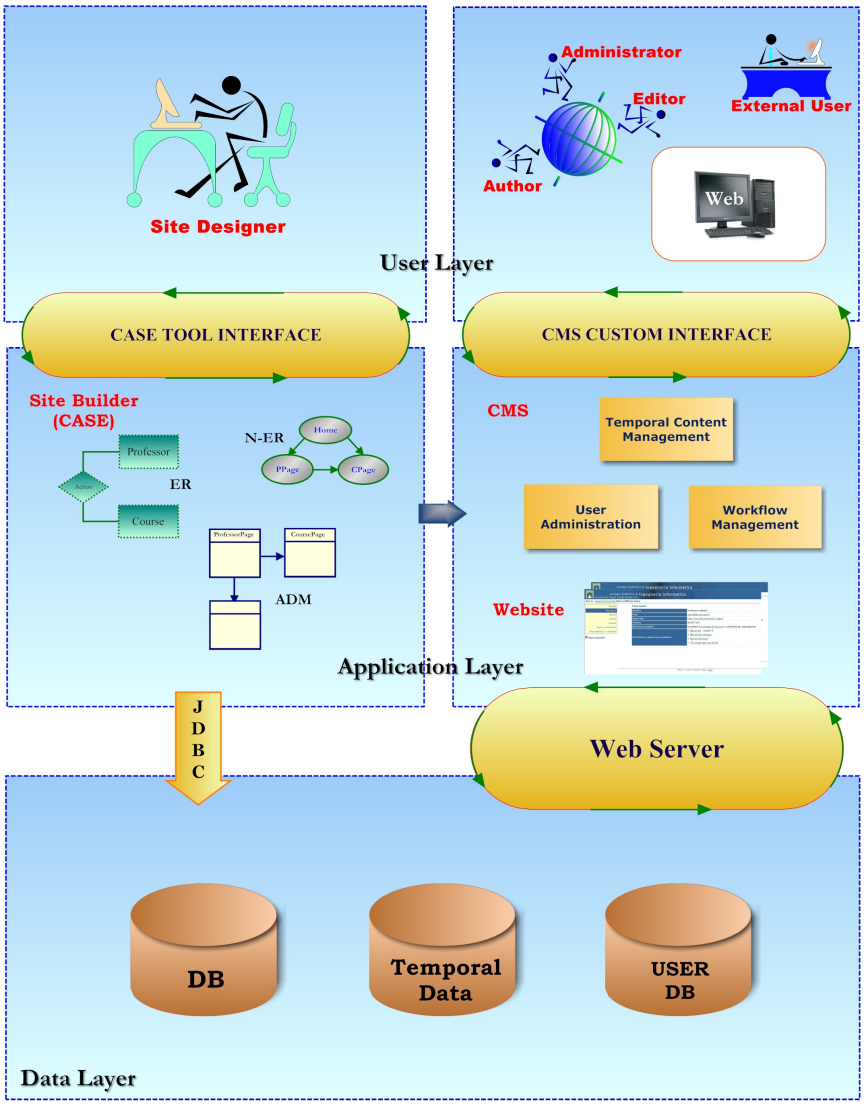


Figure 6.17: Architecture of the system.

The repository is managed through a relational DBMS (MySQL in our prototype implementation but it can be easily replaced by any other relational databases) that stores contents, temporal information and user's information. The communication with the CASE tool is realized via JDBC, while the connection with CMS (and obviously the underlying sites) is made through an Apache Web Server.

Therefore we have the Application layer that implements the aforementioned methodologies to manage contents and their temporal metadata. As we can see in the figure the elements this level includes the CASE tool through which the designer can define the structure of a site and the temporal choices for the management of the contents. On the basis of these choices the CASE tool generates the site along with the CMS for the management of the site itself.

Our tools can manage the whole content life-cycle, starting from their creation, by means of definition of the three main models (ER, N-ER and ADM through the CASE tool, see Figure 6.18 for a screenshot of the site design interface). Our CMS is fully accessible through a standard Web browser and is completed with the traditional features of workflow and user management.

The upper layer is the user layer. The CASE tool can be accessed offline by the designer of the site with a J2SE interface. With the CMS a user access and manage the contents and temporal data through a common Web browser. Each user, depending on the different role can access to the CMS functionality through a customized interface. We distinguish between internal (site designers, administrators, authors, etc.) and external (or final) users of the CMS.

6.5 An Example Application

Let us now exemplify the design process by referring to the example introduced in Section 6.2 which, despite being small, allows us to describe the main issues in the methodology. We also sketch how the tool we are implementing supports the process itself. Rather than showing a complete example, where it would be heavy to include all the temporal features, we refer to a non-temporal example, and comment on some of its temporal extensions.

The first step is to add temporal features to the snapshot ER schema. Let us assume that the requirements specify that we need: (i) to know the state (with all attribute values) of the entity *Project* when a change is applied to one or more attribute values; and (ii) to keep track of the modifications on

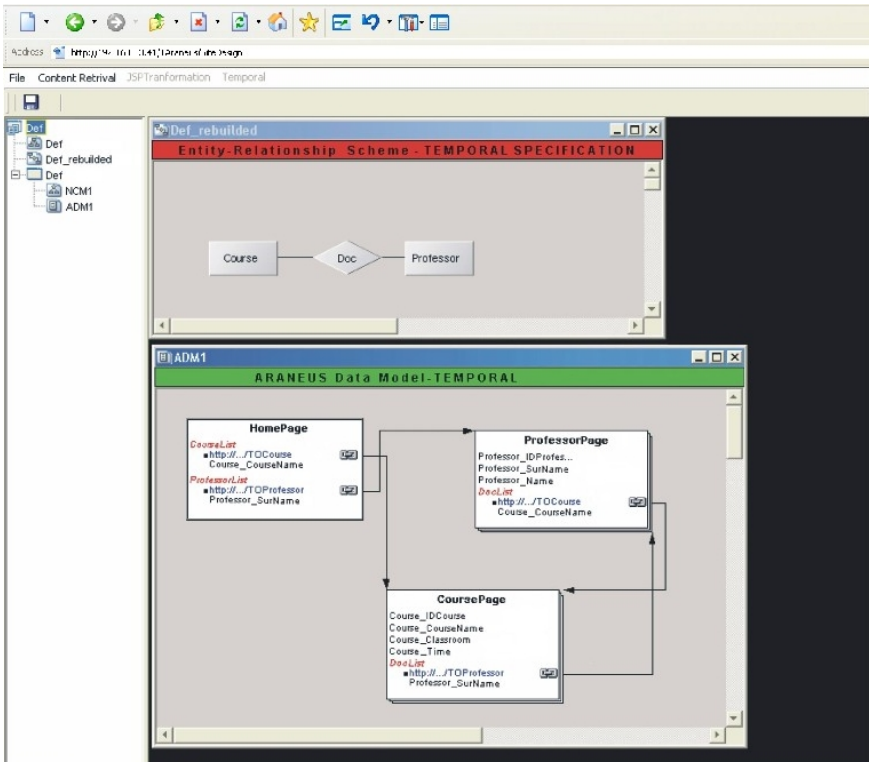


Figure 6.18: A screenshot of the CASE tool design interface.

the *office_hour* attribute for the *Teacher* entity. The first point means that the whole *Project* entity needs to be temporal, whereas for the second point, indeed, the designer has to set the temporality only for the attributes *office_hour* in the *Teacher* entity.

In Figure 6.19 we illustrate the portion of interest of the resulting T-ER schema: the elements tagged with T are those chosen as temporal (with no chronos specified).

With respect to the conceptual design of the navigation, we have already shown in Section 6.2 the overall N-ER scheme. Let us concentrate here on the temporal features: at this point it is possible to choose how to manage versions for each macroentity and attribute. We have defined the *Project* entity

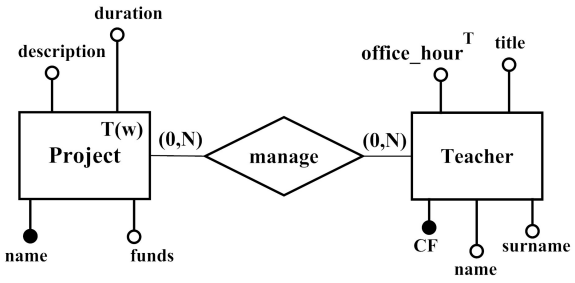


Figure 6.19: The example T-ER schema

as temporal in the T-ER model so the temporal database will handle the modifications but we don't want the site to show all versions so we choose here to have only the last version with a timestamp in the *Project* macroentity. It will be possible in the future to change this choice and add versions for projects by simply modifying this property. For the temporal attribute *office_hour* in *Teacher* we want all the versions to be managed and the snapshot attribute *title* should be modifiable via CMS. In Figure 6.20 the temporal N-ER scheme is shown with explicit indication of the version management choices (with the following codes: AV: all versions; LV: last version; SU: snapshot updatable).

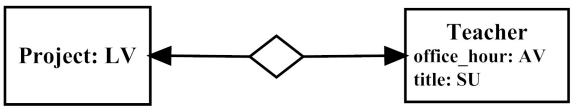


Figure 6.20: Temporal features in the N-ER model

Let us then consider the logical design. As we have discussed in Section 6.2, a standard ADM scheme can be automatically generated as an algebraic transformation based on the conceptual models (it can then be restructured if needed). During this automatic generation, for each temporal element in a page scheme, it is possible to specify how to present versions starting from the choices made in the N-ER scheme. It is also possible, in this phase, to specify whether to include meta-information and how to show it in pages. On the basis of the requirements, we could decide to handle versions for *CoursePage* by means of a CHAIN VERSION STRUCTURE. For the *TeacherPage* page scheme

we could choose to handle the last version in the main page for the *office_hour* attribute with an associated SIMPLE VERSION STRUCTURE page presenting all versions. The *Teacher* page should also present information about the last update for the *title* attribute.

The T-ADM page scheme for the *TeacherPage* is illustrated in Figure 6.21. The *office_hour* attribute is associated with the VALIDITY INTERVAL information and a LINK TO VERSION that point to the SIMPLE VERSION STRUCTURE page scheme presenting all the versions each with a VALIDITY INTERVAL. The snapshot attribute *title* has instead been associated with a LAST MODIFIED meta-information.

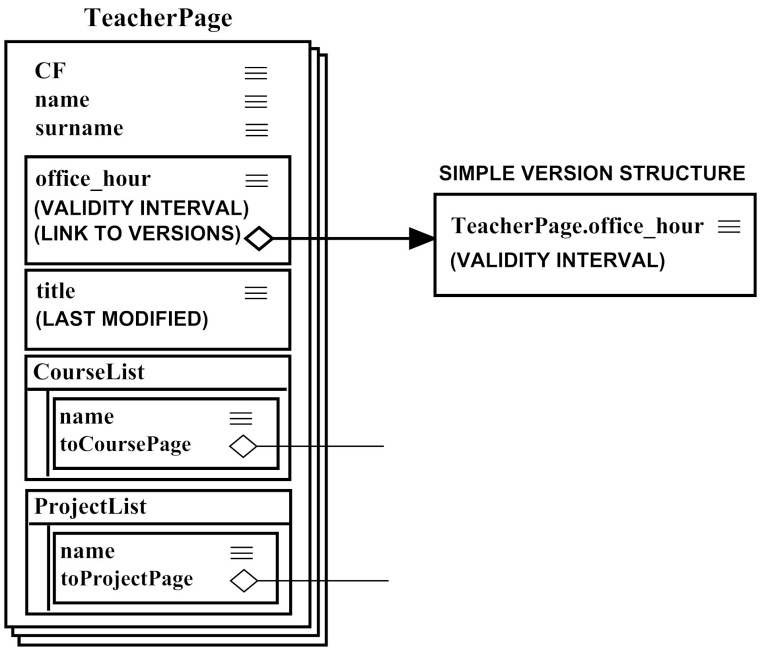


Figure 6.21: A T-ADM page scheme.

At the end of the design process, the tool can be used to generate the actual site, which can be static (that is, plain HTML) or dynamic (JSP); actually some of the features (such as the time point selector) are allowed only in the dynamic environment. It is worth noting that the sites we generate with our

approach are completely standard, as they require common http servers, with just ordinary JSP support.

The temporal Web site generation comes with the automatic generation of the CMS to manage the contents. Initially a predefined user, with administrator privileges, is created to allow the designer to define all the users/groups and rights of access to information.

6.6 Conclusions

In this thesis we have focused on one of the aspects of interest for the management of temporal evolution: the content. Other dimensions are obviously of interest for real-world, complex Web sites, and we plan to consider them in the near future. They include: (i) the presentation; (ii) the hypertext structure; (iii) the database structure. Among them, the most challenging is probably the last one: as we consider data intensive Web sites, the hypertext structure is obviously strongly related to the database structure so it could be very important to keep track of the schema evolution. If you change the ER schema (and, as a consequence, the underlying database schema), for example deleting an entity and a relationship, it can result in a change in the hypertext structure and/or the presentation. Essentially, this would be a variation of a maintenance problem, with the need to keep track of versions.

Conclusion

In this thesis, the contributions produced during my PhD, are presented. Various are the activities that have been conducted in different areas.

The main effort have been concentrated on the interoperability of Semantic Web formalisms. Here we have demonstrated the generality and the versatility of the MIDST approach, that has been extended to allow translations between Semantic Web models. The supermodel, that is at the base of the MIDST approach, was thought to work with some kind of models. Its expressivity needs to be enhanced to allow the description of the new models of interest. The two extensions produced in this thesis, regard the translation between RDF and Topic Maps and a preliminar work on the interoperability between ontologies and databases. Keeping on studying Semantic Web issues, we have then tackled the problem of storing and querying RDF data.

The relational dictionary, that MIDST uses to store schema and instance data, exploit a logical organization that perform a partition basing on the nature of the model's constructs. This kind of storage model, tuned with indexing and partitioning techniques produced results that are (almost always) better than the actual RDF storage systems, both in terms of performance and scalability.

The last topic, described in this thesis, regards the management of time in data intensive Web sites. We here produced a framework and a methodology for the modeling of the various aspects of a Web site, in particular the ones whose main purpose is the publication of data extracted from a relational database. In this context, we have introduced the design features that allow the designer to choose which are the time varying information, how to manage different versions and how to present them to the final user. The tool we have developed, automatically generate the final Web site, together with the schema of the Temporal Database and the CMS to modify contents. All the various subjects that are described in this thesis, have the common thread that is the

management of models and their interoperability, that we tackle with an high level approach, avoiding specific solutions.

Appendices

Appendix A

INITIAL SUPERMODEL CLASS DIAGRAM

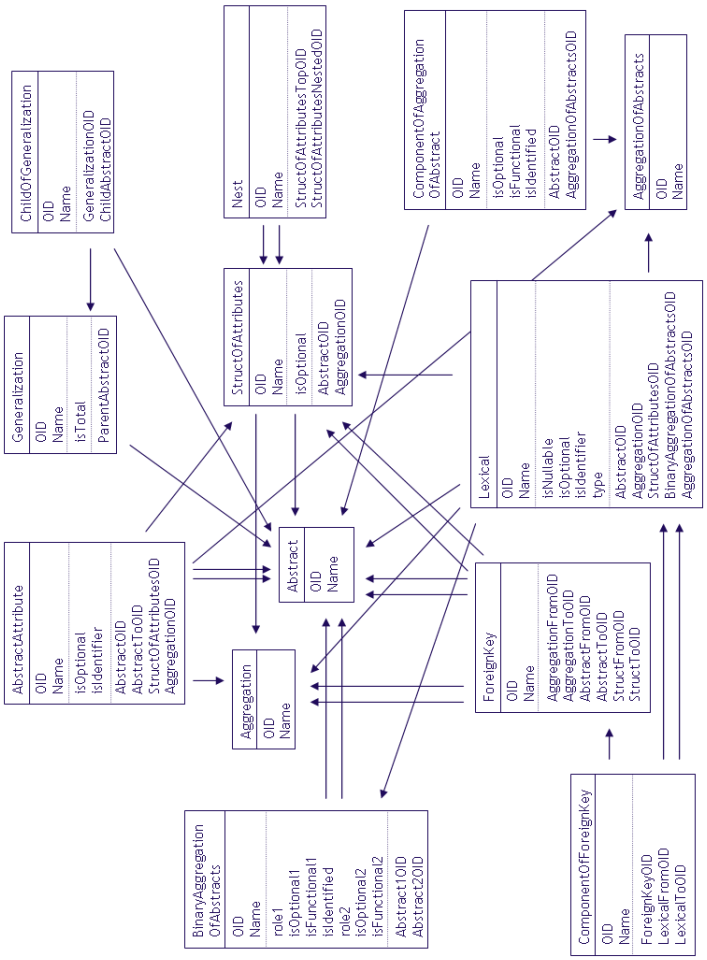


Figure A.1: The supermodel class diagram.

Appendix B

DATALOG RULES

Here we illustrate some of the representative datalog rules that we have defined for the translation RDF-TM

RDF-TM pre-rules

Pre-rules to introduce the abstracts *subject* and *object*, used as roles for Topic Maps.

```
SM_Abstract(
  OID:#CreateAbstractOID_1*("'subject'"),
  Name:"'Subject'",
  type:"'URI'"
) <-


SM_Abstract(
  OID:#CreateAbstractOID_1*("'object'"),
  Name:"'Object'",
  type:"'URI'"
) <-
```

Resource to Topic

This rule represent the one-to-one correspondence between RDF resources and Topic Maps topics.

```
SM_Abstract(
  OID:#AbstractOID_0*(absOID),
  Name:absName,
  type:type
)

<-

SM_Abstract(
  OID:absOID,
  Name:absName,
  type:type
);
```

Type to Instance Of

One-to-one correspondence between *rdf:type* and *Topic Maps* association.

```
SM_Type(
  OID:#TypeOID_0*(typeOID),
  Name:typeName,
  AbstractAsTypeOID:#AbstractOID_0(childOID),
  AbstractToBeTypedOID:#AbstractOID_0(parentOID)
)

<-

SM_Type(
  OID:typeOID,
  Name:typeName,
  AbstractAsTypeOID:childOID,
  AbstractToBeTypedOID:parentOID
```

```
);
```

Property to Association

An `AbstractAttribute`, representing an RDF property (all except for *rdf:type*), is translated into an `AggregationOfAbstracts` (representing a Topic Maps association).

```
SM_AggregationOfAbstracts(
  OID:#CreateAggregationOfAbstractsOID_1*("'abstractAttribute'",
  propertyOID),
  Name:propertyName
)

<-

SM_AbstractAttribute(
    OID:propertyOID,
    Name:propertyName
);
```

The two abstracts, corresponding to the subject and the object of the property, are translated into two `ComponentOfAggregationOfAbstracts` having the roles "subject" and "object", respectively.

```
SM_ComponentOfAggregationOfAbstracts(
  OID:#CreateComponentOfAggregationOfAbstractsOID_1*(absOID,
  "'subject'", propertyName, absToOID),
  Name:"'Subject'",
  AbstractOID:#AbstractOID_0(absOID),
  AggregationOfAbstractsOID:#CreateAggregationOfAbstractsOID_1*
  ("'abstractAttribute'", propertyOID),
  AbstractAsRoleOID:#CreateAbstractOID_1("'subject'")
)

<-
```

```
SM_AbstractAttribute(
  OID:propertyOID,
  Name:propertyName,
  AbstractOID:absOID,
  AbstractToOID:absToOID
);

SM_ComponentOfAggregationOfAbstracts(
  OID:#CreateComponentOfAggregationOfAbstractsOID_1*(absOID,
  "'object'", propertyName, absToOID),
  Name:"'Object'",
  AbstractOID:#AbstractOID_0(absToOID),
  AggregationOfAbstractsOID:#CreateAggregationOfAbstractsOID_1*
  ("'abstractAttribute'", propertyOID),
  AbstractAsRoleOID:#CreateAbstractOID_1("'object'")
)

<-

SM_AbstractAttribute(
  OID:propertyOID,
  Name:propertyName,
  AbstractOID:absOID,
  AbstractToOID:absToOID
);
```

Container to Association

An RDF *container* is translated into a Topic Maps *association*. Therefore the following rule creates an `AggregationOfAbstracts` for each `Set`.

```
SM_AggregationOfAbstracts(
  OID:#CreateAggregationOfAbstractsOID_1*("'set'", setOID),
  Name:setName
)
```

```
<-

SM_Set(
  OID:setOID,
  Name:setName
);
```

The components of the container are then translated into components of the aggregation.

```
SM_ComponentOfAggregationOfAbstracts(
  OID:#CreateComponentOfAggregationOfAbstractsOID_1*(componentOID,
  "'set'", componentName, absOID),
  Name:componentName,
  AbstractOID:#AbstractOID_0(absOID),
  AggregationOfAbstractsOID:#CreateAggregationOfAbstractsOID_1*
  ("'set'", sOID),
  AbstractAsRoleOID:#CreateAbstractOID_1("'member'")
)

<-

SM_ComponentOfSet(
  OID:componentOID,
  Name:componentName,
  SetOID:sOID,
  AbstractOID:absOID
);
```

Bibliography

[ACB06] Paolo Atzeni, Paolo Cappellari, and Philip A. Bernstein. Model-independent schema and data translation. In *EDBT*, pages 368–385, 2006.

[ACG07] Paolo Atzeni, Paolo Cappellari, and Giorgio Gianforme. Midst: model independent schema and data translation. In Chee Yong Chan, Beng Chin Ooi, and Aoying Zhou, editors, *SIGMOD Conference*, pages 1134–1136. ACM, 2007.

[ACK+01] Sofia Alexaki, Vassilis Christophides, Gregory Karvounarakis, Dimitris Plexousakis, and Karsten Tolle. The ics-forth rdfsuite: Managing voluminous rdf description bases. In *SemWeb*, 2001.

[ACT+08] Paolo Atzeni, Paolo Cappellari, Riccardo Torlone, Philip A. Bernstein, and Giorgio Gianforme. Model-independent schema translation. *VLDB Journal*, 2008. To appear, available from the first author's Web site.

[AMH07] Daniel J. Abadi, Adam Marcus 0002, Samuel Madden, and Katherine J. Hollenbach. Scalable semantic web data management using vertical partitioning. In Christoph Koch, Johannes Gehrke, Minos N. Garofalakis, Divesh Srivastava, Karl Aberer, Anand Deshpande, Daniela Florescu, Chee Yong Chan, Venkatesh Ganti, Carl-Christian Kanne, Wolfgang Klas, and Erich J. Neuhold, editors, *VLDB*, pages 411–422. ACM, 2007.

[AMM97] Paolo Atzeni, Giansalvatore Mecca, and Paolo Merialdo. To weave the web. In *23rd Conference on Very Large Database Systems*, pages 206–215, Athens, Greece, 1997.

[AMM01] Paolo Atzeni, Paolo Merialdo, and Giansalvatore Mecca. Data-intensive web sites: Design and maintenance. *World Wide Web*, 4(1-2):21–47, 2001.

[AN04] Paolo Atzeni and Pierluigi Del Nostro. T-araneus: Management of temporal data-intensive web sites. In Elisa Bertino, Stavros Christodoulakis, Dimitris Plexousakis, Vassilis Christophides, Manolis Koubarakis, Klemens Böhm, and Elena Ferrari, editors, *EDBT*, volume 2992 of *Lecture Notes in Computer Science*, pages 862–864. Springer, 2004.

[AN06] P. Atzeni and P. Del Nostro. Management of heterogeneity in the semanticweb. *icdew*, 0:60, 2006.

[APN08] Paolo Atzeni, Stefano Paolozzi, and Pierluigi Del Nostro. Ontologies and databases: Going back and forth. In *ODBIS*, pages 9–16, 2008.

[APS07] Alessandro Artale, Christine Parent, and Stefano Spaccapietra. Evolving objects in temporal information systems. *Ann. Math. Artif. Intell*, 50(1-2):5–38, 2007.

[AT07] Y. An and T. Topaloglou. Maintaining semantic mappings between database schemas and ontologies. In *SWDB-ODBIS07, Vienna*, 2007.

[BCCF06] Marco Brambilla, Stefano Ceri, Sara Comai, and Piero Fraternali. A CASE tool for modelling and automatically generating web service-enabled applications. 2:354–372, July 20 2006.

[BCF+07] Marco Brambilla, Stefano Ceri, Federico Michele Facca, Irene Celino, Dario Cerizza, and Emanuele Della Valle. Model-driven design and development of semantic Web service applications. *ACM Transactions on Internet Technology (TOIT)*, 8(1), 2007.

[Ber03] Philip A. Bernstein. Applying model management to classical meta data problems. In *CIDR*, 2003.

[BGJ06] Michael H. Bhlen, Johann Gamper, and Christian S. Jensen. Multi-dimensional aggregation for temporal data. In Yannis E. Ioannidis, Marc H. Scholl, Joachim W. Schmidt, Florian Matthes, Michael Hatzopoulos, Klemens Bhm, Alfons Kemper, Torsten Grust, and

Christian Bhm, editors, *EDBT*, volume 3896 of *Lecture Notes in Computer Science*, pages 257–275. Springer, 2006.

[BHKN06] Jennifer L. Beckmann, Alan Halverson, Rajasekar Krishnamurthy, and Jeffrey F. Naughton. Extending rdbmss to support sparse datasets using an interpreted attribute storage format. In Ling Liu, Andreas Reuter, Kyu-Young Whang, and Jianjun Zhang, editors, *ICDE*, page 58. IEEE Computer Society, 2006.

[BKvH02] Jeen Broekstra, Arjohn Kampman, and Frank van Harmelen. Sesame: A generic architecture for storing and querying RDF and RDF schema. In I. Horrocks and J. Hendler, editors, *Proceedings of the First Internation Semantic Web Conference*, number 2342 in Lecture Notes in Computer Science, pages 54–68. Springer Verlag, July 2002.

[BL98] T. Berners-Lee. Notation 3. Technical report, W3C, 1998.

[BLHL01] T. Berners-Lee, J. Hendler, and O. Lassila. The semantic web. *Scientific American*, 284(5):28–37, 2001.

[Boi] Bob Boiko. *Content Management Bible*. Wiley Publishing Inc., Indianapolis.

[CDES05] Eugene Inseok Chong, Souripriya Das, George Eadon, and Jagannathan Srinivasan. An efficient SQL-based RDF querying scheme. In Klemens Böhm, Christian S. Jensen, Laura M. Haas, Martin L. Kersten, Per-Åke Larson, and Beng Chin Ooi, editors, *VLDB*, pages 1216–1227. ACM, 2005.

[CFB+02] Stefano Ceri, Piero Fraternali, Aldo Bongio, Marco Brambilla, Sara Comai, and Maristella Matera. *Designing Data-Intensive Web Applications*. Elsevier, Amsterdam, Netherlands, December 2002.

[CGP+03] Paolo Ciancarini, Riccardo Gentilucci, Marco Pirruccio, Valentina Presutti, and Fabio Vitali. Metadata on the web: On the integration of rdf and topic maps. In *Extreme Markup Languages*, 2003.

[com03a] Lars Marius Garshol: A comparison. Living with topic maps and rdf, 2003.

[com03b] Lars Marius Garshol: A comparison. The rtm rdf to topic maps mapping: Definition and introduction, 2003.

[FFLS00] Mary F. Fernández, Daniela Florescu, Alon Y. Levy, and Dan Suciu. Declarative specification of web sites with strudel. *VLDB J*, 9(1):38–55, 2000.

[FLB+06] Tim Furche, Benedikt Linse, François Bry, Dimitris Plexousakis, and Georg Gottlob. RDF querying: Language constructs and evaluation methods compared. In Pedro Barahona, François Bry, Enrico Franconi, Nicola Henze, and Ulrike Sattler, editors, *Reasoning Web*, volume 4126 of *Lecture Notes in Computer Science*, pages 1–52. Springer, 2006.

[Gar01] Lars Marius Garshol. Topic maps, rdf, daml, oil: A comparison, 2001.

[Gar02] Lars Marius Garshol. An rdf schema for topic maps, 2002.

[Gar05] Lars Marius Garshol. A model for topic maps: Unifying rdf and topic maps. In *Extreme Markup Languages*, 2005.

[Gen02] Marco Gentilucci, Riccardo; Pirruccio. Metainformazioni sul world wide web: Conversione di formato e navigazione. Master's thesis, University of Bologna, 2002.

[GJ99] Heidi Gregersen and Christian S. Jensen. Temporal entity-relationship models - A survey. *IEEE Trans. Knowl. Data Eng*, 11(3):464–497, 1999.

[GVP+08] Giorgio Gianforme, Roberto De Virgilio, Stefano Paolozzi, Pierluigi Del Nostro, and Danilo Avola. A novel approach for practical semantic web data management. In *KES (2)*, pages 650–655, 2008.

[HG03] Stephen Harris and Nicholas Gibbins. 3store: Efficient bulk RDF storage. In Raphael Volz, Stefan Decker, and Isabel F. Cruz, editors, *PSSS*, volume 89 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2003.

[HK87] Richard Hull and Roger King. Semantic database modeling: survey, applications, and research issues. *ACM Comput. Surv.*, 19(3):201–260, 1987.

[HSV03] Siegfried Handschuh, Steffen Staab, and Raphael Volz. On deep annotation. In *WWW*, pages 431–438, 2003.

[JS99] Christian S. Jensen and Richard T. Snodgrass. Temporal data management. *IEEE Trans. Knowl. Data Eng*, 11(1):36–44, 1999.

[KOM05] Atanas Kiryakov, Damyan Ognyanov, and Dimitar Manov. OWLIM - A pragmatic semantic repository for OWL. In Mike Dean, Yuanbo Guo, Woochun Jun, Roland Kaschek, Shonali Krishnaswamy, Zhengxiang Pan, and Quan Z. Sheng, editors, *WISE Workshops*, volume 3807 of *Lecture Notes in Computer Science*, pages 182–192. Springer, 2005.

[Kri06] Madhav Krishna. Retaining semantics in relational databases by mapping them to RDF. In *WI-IATW '06*, pages 303–306, 2006.

[Lau07] Georg Lausen. Relational databases in RDF. In *SWDB-ODBIS07, Vienna*, 2007.

[MAM+00] Paolo Merialdo, Paolo Atzeni, Marco Magnante, Giansalvatore Mecca, and Marco Pecorone. HOMER: a model-based CASE tool for data-intensive Web sites. In Weidong Chen, Jeffery Naughton, and Philip A. Bernstein, editors, *Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data: May 16–18, 2000, Dallas, Texas*, volume 29(2) of *SIGMOD Record (ACM Special Interest Group on Management of Data)*, pages 586–586, pub-ACM:adr, 2000. ACM Press.

[MAM03] Paolo Merialdo, Paolo Atzeni, and Giansalvatore Mecca. Design and development of data-intensive web sites: The araneus approach. *ACM Trans. Internet Techn*, 3(1):49–92, 2003.

[MG05] Garshol L. M. and Moore G. Topic maps data model, 2005.

[MHS07] Boris Motik, Ian Horrocks, and Ulrike Sattler. Bridging the gap between OWL and relational databases. In *WWW*, pages 807–816, 2007.

[MM04] Frank Manola and Eric Miller. Rdf primer, 2004.

[Moo01] G. Moore. RDF and Topic Maps: An exercise in convergence. *XML Europe 2001*, 2001.

[MS01] Lacher M. and Decker S. On the integration of topic map data and rdf data. In *Extreme Markup Languages 2001 Conference*, Montreal, Canada, 2001.

[Nak01] R. Nakano. *Web Content Management - A Collaborative Approach.* Addison Wesley, Indianapolis, 2001.

[Ogi01a] Nikita Ogievetsky. Harvesting xml topic maps from rdf, 2001.

[Ogi01b] Nikita Ogievetsky. XML Topic Maps through RDF glasses. *Markup Languages: Theory & Practice*, 3(3):333–364, Summer 2001.

[Ont03a] Ontopia. Rtm: An rdf-to-tm mapping, 2003.

[Ont03b] Ontopia. Tmr: A tm-to-rdf mapping, 2003.

[PA07] Stefano Paolozzi and Paolo Atzeni. Interoperability for semantic annotations. In *DEXA Workshops*, pages 445–449, 2007.

[PC05] Cristian Pérez de Laborda and Stefan Conrad. Relational.OWL - A Data and Schema Representation Format Based on OWL. In *APCCM2005*, volume 43 of *CRPIT*, pages 89–96. ACS, 2005.

[PDN08] Paolo Atzeni Pierluigi Del Nostro, Stefano Paolozzi. Extending midst to semantic annotation. In *DEXA '08: Proceedings of the 2008 19th International Conference on Database and Expert Systems Application*, pages 207–211, Washington, DC, USA, 2008. IEEE Computer Society.

[PS02] David Thiemecke James Ellis Phil Suh, Dave Addey. *Content Management Systems.* Glasshaus, Indianapolis, 2002.

[RK05] Mahesh S. Raisinghani and Christopher Klassen. Temporal databases. In Laura C. Rivero, Jorge Horacio Doorn, and Viviana E. Ferraggine, editors, *Encyclopedia of Database Technologies and Applications*, pages 677–682. Idea Group, 2005.

[SLH06] Nigel Shadbolt, Tim Berners Lee, and Wendy Hall. The semantic web revisited. *IEEE Intelligent Systems*, 21(3):96–101, 2006.

[Sno99] Richard T. Snodgrass. *Developing Time-Oriented Database Applications in SQL.* Morgan Kaufmann, 1999.

[SP03] Sylvia Schwab Steve Pepper. Curing the web's identity crisis: Subject indicators for rdf, 2003.

[TA01] Riccardo Torlone and Paolo Atzeni. A unified framework for data translation over the web. In *WISE (1)*, pages 350–358, 2001.

[TBA06] Quang Trinh, Ken Barker, and Reda Alhajj. Rdb2ont: A tool for generating OWL ontologies from relational database systems. In *AICT/ICIW*, page 170, 2006.

[W3C] W3C. The resource description framework.

[W3C06] W3C. Notation 3, 2006.

[WSKR04] Kevin Wilkinson, Craig Sayers, Harumi Kuno, and Dave Reynolds. Efficient RDF storage and retrieval in jena2. Technical Report HPL-2003-266, Hewlett Packard Laboratories, January 14 2004.

[XCDS04] Zhuoming Xu, Xiao Cao, Yisheng Dong, and Wenping Su. Formal approach and automated tool for translating er schemata into OWL ontologies. In *PAKDD*, pages 464–475, 2004.

[XZD06] Zhuoming Xu, Shichao Zhang, and Yisheng Dong. Mapping between relational database schema and OWL ontology for deep annotation. In *WI '06*, pages 548–552, Washington, DC, USA, 2006. IEEE Computer Society.

[ZML+06] Jian Zhou, Li Ma, Qiaoling Liu, Lei Zhang, Yong Yu, and Yue Pan. Minerva: A scalable OWL ontology storage and inference system. In *ASWC*, pages 429–443, 2006.