

Integrating Information on the Semantic Web
Using Partially Ordered Multi Hypersets

by

Piotr Kaminski
B. Math., University of Waterloo, 1997

A Thesis Submitted in Partial Fulfillment of the
Requirement for the Degree of

MASTER OF SCIENCE

in the Department of Computer Science

We accept this thesis as conforming
to the required standard

Dr. R. N. Horspool, Supervisor (Department of Computer Science)

Dr. H. A. Müller, Supervisor (Department of Computer Science)

Dr. M. Storey, Department Member (Department of Computer Science)

Dr. I. Traoré, Outside Member (Dept. of Electrical & Computer Engineering)

© Piotr Kaminski, 2002
University of Victoria

All rights reserved. This thesis may not be reproduced in whole or in part,
by photocopy or other means, without the permission of the author.

Supervisors: Dr. R. N. Horspool and Dr. H. A. Müller

ABSTRACT

The semantic web is supposed to be a global, machine-readable information repository, but there is no agreement on a single common information metamodel. To prevent the fragmentation of this nascent semantic web, this thesis introduces the expressive and flexible Braque metamodel. Based on non-well-founded partially ordered multisets (hyper pomsets), augmented with a powerful reflection mechanism, the metamodel supports the automated, lossless, semantically transparent integration of relationship-based metamodels. Complete mappings to Braque from the Extensible Markup Language (XML), the Resource Description Framework (RDF), and the Topic Maps standard are provided. The mappings place information at the same semantic level, allowing uniform navigation and queries without regard for the original model boundaries.

Examiners:

Dr. R. N. Horspool, Supervisor (Department of Computer Science)

Dr. H. A. Müller, Supervisor (Department of Computer Science)

Dr. M. Storey, Department Member (Department of Computer Science)

Dr. I. Traoré, Outside Member (Dept. of Electrical & Computer Engineering)

Table of Contents

Abstract.....	ii
Table of Contents.....	iii
List of Tables.....	vi
List of Equations.....	vii
List of Figures.....	viii
Dedication.....	xi
1. Introduction.....	1
1.1. The Status Quo.....	1
1.2. The Semantic Web.....	2
1.3. The Case for Integration.....	3
1.4. Looking Forward.....	3
2. Background.....	5
2.1. Terminology.....	5
2.1.1. Data, Information, Knowledge and Wisdom.....	5
2.1.2. Reification.....	7
2.1.3. Uniform Resource Identifiers.....	8
2.1.4. Ontologies.....	10
2.2. Classifying Information.....	11
2.2.1. Classification of Semistructured Data.....	12
2.2.2. Metamodel Stratification.....	14
2.3. Metamodel Semantic Layers.....	16
2.3.1. IMI Reference Model.....	16
2.3.2. Object Layer Features.....	18
2.4. Mapping between Metamodels.....	20
2.4.1. Syntax-to-Syntax Mapping.....	20
2.4.2. Object-to-Object Mapping.....	21
2.4.3. Object-to-Semantic Lift.....	22
2.4.4. Semantic-to-Semantic Mapping.....	24
2.4.5. Mapping Summary.....	25
3. The Braque Metamodel.....	26
3.1. Goals and Principles.....	26
3.2. Primitives.....	28
3.2.1. Atoms.....	28
3.2.2. Hypersets.....	29
3.2.3. Identity.....	31
3.2.4. Ordering and Duplicates.....	33

3.2.5. Nest Size	35
3.2.6. Notational Sugar	35
3.3. Naïve Upper Ontology	37
3.3.1. Types	38
3.3.2. Relations.....	39
3.3.3. Membership Reification.....	41
3.3.4. Roles	42
3.3.5. Subtypes.....	46
3.3.6. Names.....	47
3.3.7. Identifiers.....	50
3.4. Inferences and Validation.....	52
3.4.1. Relation Hints	53
3.4.2. Relation Constraints	55
3.4.3. Metatype Compatibility Problem.....	57
3.4.4. Metatype Constraint	59
3.4.5. Constraining Naïve Metatypes	61
3.5. Issues of Logic	62
4. Integration	64
4.1. Extensible Markup Language.....	64
4.1.1. Basic Structure of XML Documents.....	65
4.1.2. XML Names.....	67
4.1.3. XML Namespaces.....	68
4.2. Resource Description Framework	69
4.2.1. Basic Structure of RDF.....	70
4.2.2. RDF Types, Properties and Values.....	73
4.2.3. RDFS Classes, Hierarchies and Indicators	75
4.2.4. Containers.....	77
4.2.5. Containers Embedding	80
4.2.6. Statements, Statings and Reification.....	82
4.3. Topic Maps.....	84
4.3.1. Basic Structure of Topic Maps	85
4.3.2. Subject Identification	87
4.3.3. Class Ontology.....	90
4.3.4. Scopes	92
4.3.5. Names in Topic Maps	98
4.3.6. Occurrences.....	99
4.4. Integration Example.....	101
4.4.1. Course List in RDF	102
4.4.2. Job Offerings in XML	105
4.4.3. Coverage Opinions as Topic Maps	108
4.4.4. Integrated Model in Braque	111

5. Related Work.....	114
5.1. The Integrated Metamodels.....	114
5.1.1. Extensible Markup Language.....	114
5.1.2. Resource Description Framework.....	115
5.1.3. Topic Maps.....	117
5.2. Merging RDF and XML.....	118
5.2.1. Bridging the Gap.....	118
5.2.2. OrdLab Graphs.....	120
5.2.3. The Yin/Yang Web.....	121
5.3. Merging RDF and Topic Maps.....	124
5.3.1. The Early Lifts.....	125
5.3.2. Two Semantic Mappings.....	126
5.3.3. Occurrences as Statements.....	127
5.3.4. The Syntactic Web.....	128
5.4. Other Metamodels.....	129
5.4.1. Directed Binary Graphs.....	129
5.4.2. Advanced Graphs.....	130
5.4.3. Odds and Ends.....	131
6. Conclusions.....	133
6.1. Evaluation and Contributions.....	133
6.1.1. Deep Reflection.....	133
6.1.2. Expressive Power.....	135
6.1.3. Elegance.....	136
6.1.4. Integration.....	137
6.2. Future Work.....	138
6.2.1. Theory.....	138
6.2.2. Integration.....	140
6.2.3. Implementation.....	141
6.2.4. Follow-On Work.....	142
References.....	144
A. Braque Metamodel Reference.....	153
B. Integration Metamodels Reference.....	159
C. Research Notes.....	163
C.1. Braque Background.....	163
C.2. Subclass or Instance?.....	165
C.3. Indicators and Representation.....	167
C.4. Punning on Classes.....	169

List of Tables

Table 2-1. UML's metamodel strata.....	14
Table 3-1. Kinds of nests.....	34
Table 4-1. Prefix substitutions for RDF course list	102

List of Equations

Equation 3-1. Binary relation formula shorthand	53
Equation 3-2. Binary relations consist of pairs.....	55
Equation 3-3. Transitive relation rule.....	55
Equation 3-4. Reflexive relations rule	55
Equation 3-5. Functions have only one value for each key	56
Equation 3-6. Inverse of a binary relation.....	56
Equation 3-7. Symmetric relations rule.....	56
Equation 3-8. Relating roles and indices.....	56
Equation 3-9. Rules related to nest expansion	57
Equation 3-10. Definition of metatypes	57
Equation 3-11. Extension is transitive and reflexive.....	59
Equation 3-12. The Braque metatype constraint.....	60
Equation 3-13. Intransitive relation extending a transitive one.....	62
Equation 3-14. Extension compatibility symmetry hypothesis	62
Equation 4-1. Aggregation of XML elements and attributes.....	67
Equation 4-2. XML names uniqueness constraint.....	69
Equation 4-3. Strong RDF type mapping constraint	75
Equation 4-4. All RDFS classes extend Resource	76
Equation 4-5. Translation of RDF comments and labels	77
Equation 4-6. RDF Seq embedding constraint.....	81
Equation 4-7. RDF Alt embedding constraint.....	81
Equation 4-8. Properties contain statements.....	83
Equation 4-9. RDF reification constraint.....	84
Equation 4-10. Scope identity constraint.....	94
Equation 4-11. The unconstrained scope is the bottom of the scope lattice.....	95
Equation 4-12. Two formal interpretations of theme-based scope extension	97
Equation 4-13. XHTML identification inference.....	107

List of Figures

Figure 2-1. Data pyramid	6
Figure 2-2. R. Magritte's "La Trahison des Images" (1929)	8
Figure 2-3. IMI Reference Model layers.....	17
Figure 3-1. Representation of plain and literal atoms	29
Figure 3-2. Two representations of sets	30
Figure 3-3. A recursive hyperset.....	31
Figure 3-4. M.C. Escher's "Print Gallery" (1956).....	31
Figure 3-5. Ordered members.....	34
Figure 3-6. Labelling elements to externalize their identity.....	36
Figure 3-7. References and scoping	36
Figure 3-8. Equivalent representations of membership.....	37
Figure 3-9. Binary relationship shorthand	37
Figure 3-10. Booleans class	38
Figure 3-11. Booleans is a classifier	39
Figure 3-12. Classifiers is an instance of itself.....	39
Figure 3-13. Example of a binary relation	40
Figure 3-14. Typed relationship shorthand	40
Figure 3-15. Example of binary relationship shorthand	41
Figure 3-16. Example of relationships tagged with roles.....	44
Figure 3-17. Membership role-playing shorthand	44
Figure 3-18. Unordered binary role-playing relationship shorthand.....	45
Figure 3-19. Example of role-playing relationships using compact notation	45
Figure 3-20. Roles played by members of Play Role by Index relators.....	45
Figure 3-21. The Extend relation	46
Figure 3-22. Extend relationship example and shorthand	46
Figure 3-23. Top of the inheritance hierarchy: Ideas, Nests and Classifiers	47
Figure 3-24. Relations inheritance hierarchy	47
Figure 3-25. A name with scoped representations.....	48
Figure 3-26. Simple naming example and shorthand.....	49
Figure 3-27. Naming the Denote relation.....	49
Figure 3-28. A model of the names of "A-Sitting On A Gate"	50
Figure 3-29. A hierarchy of indication relations.....	51
Figure 3-30. Example of identification by URI	52
Figure 3-31. Example of relation hints.....	54
Figure 3-32. Example of incompatible metatypes.....	57
Figure 3-33. Example of valid cross-metatype inheritance.....	58
Figure 3-34. Metatypes inverting instances' extension.....	59
Figure 3-35. Expand relationship shorthand	61
Figure 3-36. Improved model of naïve relations	61

Figure 4-1. Mapping of sample XML document	66
Figure 4-2. Sample XML element and attribute types	67
Figure 4-3. Four kinds of XML names.....	68
Figure 4-4. Top level XML namespaces	68
Figure 4-5. Local attribute XML namespaces	69
Figure 4-6. XML namespaces.....	69
Figure 4-7. Sample RDF model in NTriples format.....	71
Figure 4-8. Mapping of sample RDF model	73
Figure 4-9. Basic RDF ontology embedding	74
Figure 4-10. RDFS classes embedding.....	76
Figure 4-11. RDFS properties embedding.....	77
Figure 4-12. Repeated statements create an implicit collection	77
Figure 4-13. Mapping of repeated statements.....	78
Figure 4-14. RDF container class hierarchy.....	78
Figure 4-15. RDF membership properties model.....	79
Figure 4-16. Explicit container holds a collection	80
Figure 4-17. RDF containers embedding	80
Figure 4-18. Braque model of explicit RDF container	81
Figure 4-19. Sample topic map in LTM format	86
Figure 4-20. Mapping of topic map sample into Braque	87
Figure 4-21. Example of subject identifiers and indicators.....	89
Figure 4-22. Mapping of identifiers and indicators into Braque	90
Figure 4-23. Specifying the type of a subject using an explicit association.....	90
Figure 4-24. Shortcut for specifying the type of a subject	91
Figure 4-25. Specifying a generalization association	91
Figure 4-26. Topic Map classification and generalization embedding.....	92
Figure 4-27. Example of association scoping.....	93
Figure 4-28. Embedding of Topic Maps scopes	94
Figure 4-29. Example mapping of scopes into Braque.....	95
Figure 4-30. Sample results of two interpretations of scope extension.....	97
Figure 4-31. Assigning base and variant names to subjects	98
Figure 4-32. Mapping Topic Maps names to Braque	99
Figure 4-33. Sample occurrences.....	100
Figure 4-34. Occurrences relation and roles	100
Figure 4-35. Sample mapping of occurrences into Braque.....	101
Figure 4-36. RDF course list fragment	103
Figure 4-37. Course list schema in Braque.....	104
Figure 4-38. IKM course in Braque.....	104
Figure 4-39. HCI course in Braque	105
Figure 4-40. XHTML job offerings example.....	106
Figure 4-41. XHTML schema fragment in Braque	107
Figure 4-42. XHTML model fragment in Braque.....	108

Figure 4-43. Coverage opinions topic map.....	109
Figure 4-44. Topic map definitions in Braque.....	110
Figure 4-45. Topic map associations in Braque.....	111
Figure 4-46. Essentials of the integrated model in Braque.....	112
Figure 5-1. A ternary relationship in RDF.....	116
Figure A-1. Braque metamodel graphical notation primitives.....	153
Figure A-2. Braque metamodel graphical notation shorthands.....	154
Figure A-3. Braque metamodel graphical notation relation hints.....	155
Figure A-4. Domains and basics of the naïve upper ontology.....	156
Figure A-5. Naïve upper ontology relations.....	157
Figure A-6. Naïve upper ontology roles.....	158
Figure B-1. XML mapping ontology.....	159
Figure B-2. RDF and RDFS classes.....	160
Figure B-3. RDF and RDFS properties.....	161
Figure B-4. Topic Maps mapping ontology.....	162
Figure C-1. G. Braque’s “Still Life with Violin and Pitcher” (1909-10).....	164
Figure C-2. Example representation of HTTP GET result.....	168
Figure C-3. Datatype punning in RDF.....	169
Figure C-4. Class and role punning in Topic Maps.....	170

*To Kate,
for pretending to be interested.*

1. Introduction

Ubiquitous networking has dramatically changed the face of computing. Most computers now participate in the global Internet and the tasks we expect them to perform have changed accordingly. Their role in supporting human communications has grown increasingly important, to the point that it has become an arguably critical part of the world's social and economic frameworks.

1.1. The Status Quo

Email and the web are today's most commonly used communication tools. They share an important property: under normal circumstances, their content—be it messages or web pages—can only be understood and acted on by humans. This is because the information communicated through these media can only be extracted with an understanding of the language used and, increasingly on the web, the visual arrangement of the content, including any embedded images. While computers enable the exchange of information between humans, they serve merely as a conduit for the data.

The Internet's users are slowly becoming aware that this lack of "understanding" by computers is severely limiting the network's utility as an information resource. Searching for information is inefficient, since web search engines can only index data with just a rudimentary grasp of the import of a web page provided by natural language processing routines. Different web directories organize their listings in different ways and, with no automated way to map between the structures, finding corresponding categories in different taxonomies requires significant effort. Collating information on one subject from multiple web pages is a painstaking manual process, since computers are not aware that web pages may describe individual items and that some of these items may be the same.

A number of initiatives that attempt to address the issue are under way. Even before the web phenomenon began, some industry sectors managed to agree on

common information exchange vocabularies within their limited vertical domain. With the advent of the Extensible Markup Language (XML) [BPS00], these efforts mushroomed into a veritable jungle of domain-specific lexicons that are successfully being used by specialized applications. However, because XML is nothing but a structured data exchange syntax, each standard encodes the domain's concepts in a different way. This makes it impossible to build domain-independent tools to integrate and process the information.

1.2. The Semantic Web

The proposed semantic web vision,¹ as proselytized by Tim Berners-Lee et al. [BHL01], is intended to provide a solution to these problems. Whereas the web as we know it can be thought of as an ocean of pretty, linked, human-readable data islands, and XML vocabularies provide certain archipelagos with specific and incompatible dialects, the semantic web proposes to create a global sea of rich machine-comprehensible information.

While not universal, the semantic web vision has many enthusiastic adherents; what is not clear is the concrete path that would lead to the realization of the vision. The World Wide Web Consortium (W3C) is promoting the Resource Description Framework (RDF). The International Standards Organizations (ISO) is developing Topic Maps and other less well-known standards. An ever-growing multitude of other approaches can be found, some aiming explicitly at the semantic web, others unaware that they are playing in this highly contested arena. Finally, we must not forget the silent majority that is quite happy with publishing data in some specialized dialect of XML or for human eyes only: the information trapped in their documents would make a great contribution to the semantic web.

¹ See [For02] for a utopian version of the vision, and [Doc01] for a dystopian account.

1.3. The Case for Integration

It is my contention that no single semantic web standard is likely to gain wide held acceptance, as each plays to a different audience, providing them the advantages they seek with a minimum of extra baggage. Combining the many markup languages into one and forcing all authors to use it is not socially viable. It may also be technically undesirable since the additional complexity introduced would not be acceptable for simpler applications. Even with their relatively low complexity, RDF and Topic Maps are already a hard sell.

So it seems that we are back where we started, except that, instead of having incompatible data exchange formats, we have incompatible information exchange formats. It would undoubtedly be valuable to be able to integrate information from different sources [Euz02]. An integrated information base would allow uniform queries over information from all sources, and greatly expand the amount of knowledge that could be inferred from the combination of models. Yet it turns out that none of the proposals currently on the table is powerful enough to elegantly integrate information expressed using other approaches.

1.4. Looking Forward

My contribution is an elegant conceptual metamodel called Braque. It is demonstrably a superset of the most popular proposed semantic web metamodels and arguably sufficiently powerful to encompass all future approaches within its framework. I also provide a proof-of-concept implementation used to showcase an example of information integration.

The rest of this thesis is laid out as follows:

- Chapter 2 gives background information about the semantic web research area, explains some of the more abstruse terminology, and sets the framework within which metamodel integration approaches can be evaluated.

- Chapter 3 introduces the new Braque metamodel and its graphical notation, an integration-friendly upper ontology, and some logical constraints on model validity.
- Chapter 4 explains the XML, RDF and Topic Map standards, specifies mappings to the Braque metamodel, and provides a concrete example of model integration.
- Chapter 5 explores related work, including other metamodels, mapping and integration techniques, and compares them to Braque.
- Chapter 6 concludes this thesis by evaluating the work presented in this thesis, summarizing my contributions, and looking at potential areas of future research.
- Appendices A and B provide reference diagrams for Braque and the three mappings, matching chapters 3 and 4, respectively.
- Appendix C holds assorted research notes, dealing in more detail with certain topics not directly relevant to the core focus of this thesis.

2. Background

Semantic web research lies at the confluence of a number of other research streams. It borrows from work in databases, artificial intelligence, distributed systems, information theory and philosophy, among others. Such a heterogeneous ancestry can make the material hard to understand, since few people possess expertise in all of the above areas.

This chapter clarifies the terminology and explains some of the concepts and technologies underlying the field. It also lays out a framework for describing and comparing semantic representation and exchange mechanisms.

2.1. Terminology

This section introduces the basic terms and concepts relevant to the semantic web.

2.1.1. Data, Information, Knowledge and Wisdom

The terms data, information, knowledge and wisdom are closely related and often confused in the English language. We shall adapt the following definitions commonly accepted in Artificial Intelligence; the examples are taken from [How01].

- Data: Symbols such as numbers, characters or images that stand only for themselves. For example, 1234567.89 is data.
- Information: Data interpreted in a context that gives it meaning, making it into facts. For example, "Your bank balance has jumped 8087% to \$1234567.89" is information.
- Knowledge: Information together with inference rules that can be used to derive new knowledge. For example, "Nobody owes me that much money" is knowledge.

- Wisdom: Knowledge applied to a decision-making process, resulting in actions. For example, "I'd better talk to the bank before I spend it, because of what has happened to other people" is wisdom.

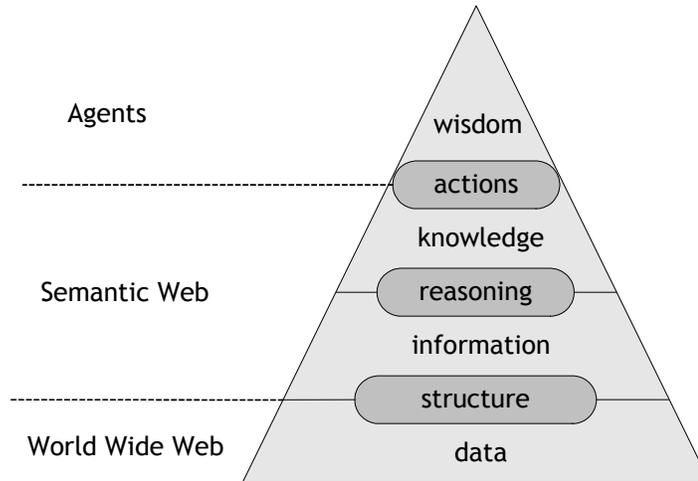


Figure 2-1. Data pyramid

From the point of view of humans, the World Wide Web holds information. Unfortunately, from the point of view of computers, the vast majority of it is only meaningless data. While it is not impossible for a client to semi-automatically extract information out of the data (for example, scraping it out of HTML as in [Kal02]), it is not easy, and often involves unreliable natural language processing techniques. The conversion parameters must also be updated manually every time the format of the data changes, which happens all too often on the visual design-oriented web. Due to the huge amounts of data currently available on the web semi-automatic extraction will be very important in the short term. The eventual goal, though, is to have authors reliably convert their own data into information and publish the information directly in a computer-understandable format.

This pool of interconnected information will be our Semantic Web. Computers will be able to “understand” this information—that is, manipulate it at a more abstract level to achieve results that rely on the information’s meaning. This kind

of manipulation, including queries and inferences, will turn the information pool into knowledge for the controlling client. The client could be a human, or it could be a software agent, making decisions on behalf of its master. While the specifics of agent decision-making are outside the scope of this thesis, agents could well become the primary users of the semantic web and hence their needs with respect to knowledge access and manipulation were considered at par with the humans' while working on this project.

2.1.2. Reification

To be pertinent and useful, the Semantic Web must contain information about real-life objects, events and ideas. Since it is impractical to store and transmit physical items in a computer network, and impossible to do the same for non-physical ideals, we will need to find a way to represent them using the resources at our disposal. This process is called reification.

Reification is defined as regarding something abstract as a material thing [Mil01]. In the context of modeling it means to create a representation within the model itself of something that is outside of the model; in other words, to make the external thing "real" within the closed world of our system. Humans have done it from times immemorial: pictures reify things within the world of the written page, spoken words reify things within the world of oral communication, and ideas reify things within our own minds. A picture of a pipe is not the pipe itself (Figure 2-2, "This is not a pipe."), but we understand that it reifies one and while looking at it can make statements about the actual object without needing to have it in front of us.

I was unable to obtain an acceptable copyright license for this picture for purposes of web publication. Please obtain a printed copy of this thesis from the University of Victoria library, or try searching the web for [Magritte's La Trahison des Images](#).

Figure 2-2. R. Magritte's "La Trahison des Images" (1929)
© Estate of René Magritte / SODRAC (Montréal) 2002

A model that uses reification consists of two things:

1. A formal system of symbols that can be manipulated.
2. An interpretation that relates the model's symbols to the real things being modeled.

While Uniform Resource Identifiers have been widely adopted as the reifying symbols, there is little agreement on how to organize and manipulate them, and how to interpret the resulting structures.

2.1.3. Uniform Resource Identifiers

Uniform Resource Identifiers (URIs) are defined in [BM+98a] as "compact strings of characters for identifying an abstract or physical resource". They are commonly used to address pages on the web (<http://www.idealnest.com>), specify FTP locations (<ftp://seng330@www.engr.uvic.ca/notes.txt>) and name newsgroups (<news://news.uvic.ca/uvic.engr.seng330>). Their ubiquitous acceptance stems from the fact that they are human readable and can be easily extended to encompass any number of protocols without compile-time changes to existing tools thanks to a consistent yet flexible syntax. Only new protocol handlers may need to be plugged in; the generic URI parser never changes.

Officially, URIs are meant to identify resources. A resource, in a somewhat circular definition, can be anything that has an identity. This of course includes electronic documents and services, but also covers concrete physical items, humans, abstract ideas, etc. Note however that the entity returned when a URI is dereferenced (using a web browser, for example) is generally *not* the resource itself; rather, it is some representation of it, or any other related entity that the resolution service thought appropriate to return. This formal extra level of indirection often conflicts with a purely intuitive interpretation of URIs.

The impressive momentum of URIs is impossible to ignore, so just about all semantic web proposals use URIs as their symbolic identifiers; this is both a blessing and a curse. On the one hand, it means that there is some hope of integrating the various systems since they use the same basic symbols and thus may even share some vocabularies. On the other hand, URIs have so long been used only for dereferencing through browsers that few people have bothered to officially define the identified resource. This can result in considerable disagreement over what a given URI identifies.² For example, does <http://www.jeep.com> identify:

- the Jeep company;
- the Jeep series of cars;
- the Jeep web site;
- the Jeep home page;
- the index.html file (or equivalent) that is returned by the web server;
- the web server itself?

When browsing the web, the answer does not really matter since the reader only cares about the web page that is displayed when he or she types in the address. In the semantic web, having everybody agree on an answer is paramount since

² Actually, there is not even agreement on whether a URI's scheme limits the kind of resources it can identify. For example, can "http:" URIs identify anything, or just documents?

people will be making formal statements about the identified resource (“<http://www.jeep.com> is great!”) and may never actually dereference the URI. Even if everybody agrees on what resource a URI identifies at a given time, definitions tend to drift or be changed, and not everyone is notified of or agrees with the changes. If an agent tries to merge information coming from sources that disagree on the interpretation of the URI, and it blindly assumes that each URI identifies a unique resource³, it will at best end up with nonsense or at worst with statements that will be misinterpreted relative to their authors’ intentions.

2.1.4. Ontologies

In its simplest form, an ontology is a dictionary that gives each symbol a definition in some well-known language with generally agreed-upon semantics (for example, English). Even if the definitions are not perfectly unambiguous (as may well be the case in English), they still provide a starting point preferable to the interpretive free-for-all described in the previous section. If two documents explicitly declare that they draw their symbols from the same ontology (or ontologies), it is reasonably safe to merge their statements since the authors at least intended to refer to the same concepts.

An ontology can define any kind of symbols. Some ontologies will define well-known individual concepts (e.g., the Jeep company), while others may define classes of individuals (e.g., the class of companies). The instances can then be related to their classes using standard “instance of” relationships defined in yet another ontology.

Ontologies intended for public use are already being published on the web. For example, the Dublin Core Initiative (<http://www.dublincore.org/>) is labouring on standardizing document metadata terms that define the characteristics of a work, such as its title, creator, coverage, or publication date. The Creative Commons is

³ This appears to be a reasonable assumption since the standard [BM+98a] clearly states that URIs are unambiguous.

building a standard vocabulary for describing copyright license terms (<http://creativecommons.org/metadata/spec>). WordNet [Mil01] is the ontological equivalent of an English thesaurus. The less formal Friend of a Friend (FOAF) vocabulary (<http://xmlns.com/foaf/0.1/> and [Dum02]) covers concepts needed to describe professional collaboration, such as people and their relationships, organizations and projects. For domain-neutral concepts, such as *entity*, *relation* and *part-of*, the IEEE is designing a Standard Upper Ontology (<http://suo.ieee.org/>) that tries to merge the many upper ontologies deployed as part of existing applications.

What distinguishes an ontology from any run of the mill semantic web document? First, top-level ontologies need to have out-of-band definitions for their terms so as to semantically bootstrap the system. Second, to be useful, an ontology's definitions should be used in many other documents; in a usage graph, ontologies are likely to be closer to the center, while other documents will tend to collect at the periphery. However, the distinction is fuzzy, since any document whose terms are referred to by others becomes a de facto ontology.

Finally, ontologies can be more or less useful depending on the subject area covered and the precision and stability of the definitions. As the semantic web takes off, some of its proponents expect that an ecology of ontologies will evolve and a form of natural selection will trim down the field to a reasonable size. For any remaining overlapping ontologies people will construct mappings between the terms according to some (hopefully) standard translation ontology.

2.2. Classifying Information

To impose order on a chaotic world, humans have been classifying objects since ancient times (see for example Genesis 2:19-20). A class abstracts away the irrelevant details of its individual members, keeping only the essential shared core. Classification lets us make sweeping generalizations, augmenting our expressive power and clarifying the perceived structure of our world. While it is possible to

make do without classes (see the programming language Self [US87], for example), classification is so familiar that few knowledge representation systems choose to go without. Class hierarchies are also the most popular content of ontologies, to the point that some people conflate ontologies with taxonomies (classes).

This section gives a quick overview of the most important distinctions between classification systems, characterizes the kind of data that is likely to appear on the semantic web, and explains the impact on the choice of a classification method. It then elucidates the initially tricky subject of stratified type models and gives the definition of *metamodel* as used in this dissertation.

2.2.1. Classification of Semistructured Data

Classical metamodels (the relational data metamodel, for example) assume that all data to be stored fits some particular, and often quite detailed, schema. This schema must be defined in full before data can be input, and each record must match the schema's constraints precisely. Schema evolution is often painful since, even if the particular database management system has good support for it, chances are that the applications connecting to the database were written strictly for the original schema and will need to have many of their assumptions re-examined.

This highly structured data model works well when the organization of data is well known ahead of time and a central authority has full control of it. Unfortunately, this situation is the antithesis of what actually happens on the web. The web (semantic or otherwise) is decentralized. No one can impose a uniform schema on the data, and even should one be adopted coordinated evolution is nigh impossible. Structured data models are a bad fit for the web.

Yet the data is not completely without structure. Some authors do follow the same schemas (more-or-less), and even schema-less data is often fairly regular. This is semistructured data – and, by extension, semistructured information – on

the semantic web. We obviously need a flexible database to store this data at all. To really take advantage of the information contained therein, though, we need to identify as much structure as possible in the raw data. This is the domain of classification.

The basic idea of classification is to assign (reified) objects to classes, so that we can treat all the instances uniformly in some respect. The first question, then, is how are objects assigned to a class. While a survey of data pattern identification and matching algorithms is outside the scope of this thesis, it is useful to note the degree to which the algorithm can be integrated into the database. The classes can either be assigned externally, by naming the object and the class it belongs to, or implicitly, by defining a class using some predicate that accepts or refuses objects based on their properties.

Next, we must ask how many classes an object can be assigned to, and whether once made, the assignment is frozen or can be changed. Most class-based programming languages only support single, static classification: each object is an instance of precisely one class, and this class is fixed at the time the object is created. In an information model, where objects are usually longer-lived and our understanding of their properties likely to evolve with time, multiple dynamic classification is preferable. This allows an object to be assigned to any number of classes simultaneously, and the assignment to be changed at any time.⁴

Finally, systems that support classes typically also support a standard binary inheritance relationship between them (also known as specialization or subtyping). The extent of support for inheritance varies along the same lines as classification: single or multiple, static or dynamic. As expected, multiple dynamic inheritance is preferable since it allows the class structure to adapt as closely as possible to the semistructured data.

⁴ Systems that sport internal, predicate-defined classes must perforce support multiple dynamic classification.

2.2.2. Metamodel Stratification

In some systems, the objects are divided into strata based on classification. Each object belongs to exactly one stratum, and all objects in a stratum must be instances of classes in the next higher stratum.⁵ Those classes, considered as objects in their own right, are in turn instances of metaclasses in the next stratum, and so on. (The prefix “meta” is commonly used to indicate a rise of one stratum.) If the meta-tower is finite, we eventually reach the top stratum where objects are not instances of any classes; they just *are*. No relationships beside classification are allowed across strata.

The Unified Modeling Language (UML, [OMG01]), for example, employs a four-layer⁶ stratified metamodel. The definition of the layers is detailed in Table 2-1 (adapted from [OMG01]). Objects in M_n are instances of classes in M_{n+1} , for $0 \leq n \leq 2$. Objects in M_3 , the top layer, are not classified; they are completely self-defined.

Layer / Stratum	Description	Examples
M_3 : meta-metamodel	The infrastructure of the metamodeling architecture.	MetaClass, MetaAttribute, MetaOperation
M_2 : metamodel	Objects that specify the language for defining the model.	Class, Attribute, Operation, Artifact
M_1 : model	Objects that specify the structure of the information domain; user-defined classes.	Person, name, writeThesis, Dissertation
M_0 : user objects	Objects that represent individuals; most of the actual data resides here.	<com.ideanest.Person 14218>, “Peter”, void writeThesis(), <Dissertation final copy>

Table 2-1. UML’s metamodel strata

⁵ In principle, a metamodel could be stratified according to any other intransitive relation, but in practice there appears to be little use for such variations. At most, strata are defined based on specializations of the classification relation, as in [AO+02].

⁶ The words “layer” and “stratum” are used interchangeably in the literature. A stratified metamodel is said to follow a fixed layer metamodeling architecture.

The alternative to a stratified metamodel is an unstratified one, where classes and their instances are intermixed and can be directly related. Classification, though it may be a privileged relationship, does not partition the model. A trademark characteristic of unstratified metamodels is that they do not arbitrarily cap the classification hierarchy. Instead, the “most meta” class is an instance of itself, forming a classification loop that provides a finite representation of an infinite classification tower.

RDF Schema (RDFS, the schema component of the Resource Description Framework, RDF [LS99]) is a recent example of an unstratified metamodel. This has proved to be a mildly controversial choice; it seems that some people strongly favour stratified metamodels. The arguments advanced against non-fixed layer metamodeling architectures include (see [PH01a], [NWC00] and RDF mailing list discussions):

- Unstratified metamodels are non-standard and difficult to understand.
- If classes are considered as sets, then having a set be a member of itself may break the Foundation Axiom of set theory and is suspiciously close to the Russell Paradox.⁷
- The semantics of unstratified metamodels are difficult to formalize and lend themselves to the “layer mistake” problem (though how one can make a “layer mistake” when there are no layers is not made clear).

Of course, there are corresponding counter-allegations from the other side of the fence. The killer argument, though, is that stratified metamodels can be embedded into unstratified ones with the addition of a few extra constraints, while the converse is not true. Accordingly, Braque—the integration-friendly metamodel

⁷ The Russell Paradox [Kle01] is a variation on the Liar’s Paradox that goes as follows: consider the set of all sets that do not contain themselves—is the set a member of itself or not? Standard axiomatic set theory avoids the issue by forbidding this kind of set from being defined.

introduced in this thesis—is unstratified, so that it can be the target of natural mappings from both varieties of metamodels.

In an unstratified model, it does not make sense to use the prefix “meta” to refer to specific layers. Thus, for the remainder of this thesis, the word “metamodel” will be used to loosely signify the domain-independent mechanics (layers M_2 and M_3 above), and the word “model” the domain-specific objects and classes (layers M_0 and M_1 above), for both stratified and unstratified architectures.

2.3. Metamodel Semantic Layers

The Information Model Interoperability (IMI) Reference Model presented in [MD00] describes a generic layered architecture that can be superimposed over most metamodels. While the IMI proposal is new and unproven, and has not yet gained widespread acceptance, it provides a useful framework for comparing metamodels and investigating possible mappings at a higher level of abstraction. This section is a summary of the relevant portions of [MD00], with a few criticisms and opinions thrown in for good measure.

Note that the layers presented here are orthogonal to the classification strata described in Section 2.2.2. Objects and classes across all strata belong to the object layer in the IMI Reference Model.

2.3.1. IMI Reference Model

The IMI Reference Model takes its cue from the well-known, widely adopted and successful OSI network model. It follows a standard layered architecture where each layer provides increasingly more abstract services to the layer above (see Figure 2-3, as presented in [MD00]).

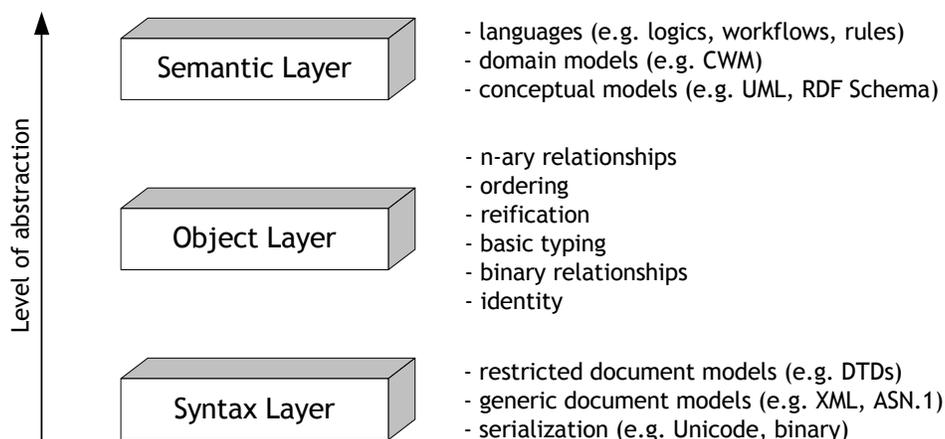


Figure 2-3. IMI Reference Model layers

Starting at the bottom, the syntax layer is only concerned with serializing information into a low-level data format from which the information can later be recreated. While a detailed and complete specification of a model's syntax layer is crucial when defining an interchange standard, in the larger scheme of things the actual syntax chosen is mostly irrelevant. Some formats may be easier to write by hand, others may be easier to parse, some (like XML) may be neither, but in the end the data at this level is only a temporary representation and will rarely be operated on directly (see Section 2.4.1 to understand why).

The object layer in the middle provides an object-oriented view of the information to the client application. It provides direct access to the basic features of an information model (discussed below in Section 2.3.2), such as identity, binary relationships and classification. It is primarily at this level that various information models are differentiated.

The top semantic layer is responsible for mapping the information primitives of the layer below to the real world (e.g., with a model theory). This layer also holds more advanced knowledge-level facilities, such as standard ontologies and inference languages. "The ultimate goal of the Semantic Web is to make applications interoperate in the semantic layer." [MD00]

Of course, one person's object layer may well be another's syntax layer. For example, XML intrinsically defines identity, basic typing and ordering and, with the addition of XLink [DMO01], it covers binary and n -ary relationships as well. Some choose to use these features directly to represent information objects; others merely use them as a well-defined lower-level mechanism for expressing their own object layer model [PS02]. Though sharing a common language on the surface, these two uses are mutually incompatible and should not be confused. Section 2.4.3 delves deeper into this problem.

2.3.2. Object Layer Features

The object layer encompasses the basic structural features of a metamodel. Not every feature in this layer is necessarily primitive—some may be defined with the help of an adjunct ontology—membership is a function of how important a feature is for modeling.

The IMI Reference Model assumes that all metamodels subscribe to the “objects and relationships” view of the world (for an alternative, see Section 6.2.1). As such, it is imperative that every metamodel be able to represent unique objects (*identity*) and simple links between them (*binary relationships*). Each object has its own identity; the notion of value equality (if any) is distinct, and does not affect identity. The identity key may be hidden, but when models are exchanged it must be externalizable in some common format (e.g., URIs). The binary relationships are simple directed links between objects and should not be confused with associations, which describe classes of links and are not included in this layer. Groups of binary links of one type from one object to many targets are also used to model collections.

Given that classification is such a common and important activity (see Section 2.2), it should come as no surprise that the IMI Reference Model expects most metamodels to provide *basic typing* facilities. Each object should be assigned a

type, also represented by an object.⁸ Nothing more is required from the type model—type specialization is notably absent—and both stratified and unstratified styles are permitted.

Next up is link and association *reification*.⁹ The idea here is simply to make links and associations (link types) first-class citizens of the model. Once a link or association is seen as an object, it can be further referenced in other links to provide further details about it. For example, link reification is often used to indicate the provenance of links, which can then be a factor when trying to ascertain the truth of the implied statement. In general, reification is a powerful feature that greatly increases a metamodel's flexibility.

When an object is involved in many links, especially of the same type, its participation in the relationships may need to be *ordered*. For example, if an object representing a research paper is linked to multiple authors, we need to model the order in which the authors are listed. Similarly, if an author is linked to all her research papers, we may wish to order the links chronologically, or in order of perceived importance. Link ordering is often implemented with the use of a standard ordering ontology, but it is nonetheless a basic feature necessary to model many common (if subtle) situations.

The final component ascribed by [MD00] to the object layer is *n-ary relationships*. An *n-ary* relationship is logically equivalent to a link between *n* objects, though they are often composed from a standard arrangement of binary links. The authors do not make clear whether the objects participating in the relationship are merely ordered or if the role that each plays is explicitly indicated. Depending on the modeling style adopted, *n-ary* relationships may not be used very often, but they are difficult to emulate properly if the feature is not available.

⁸ This seems to implicitly require type reification, excluding metamodels where types fall completely outside the model. It is not clear whether the authors intended to impose this restriction or if it's accidental.

⁹ *Reflection* would probably be a better term for this mechanism, but at this point *reification* is too entrenched to be displaced.

2.4. Mapping between Metamodels

When integrating disparate models, there must be some kind of mapping between the elements of the source and those of the unified target. The mapping can be performed at different levels, described by the IMI Reference Model. This section explains some potential information integration styles and their tradeoffs in terms of the reference model's layers, with analogies to natural language translation.

2.4.1. Syntax-to-Syntax Mapping

At first glance, the simplest approach to model interchange might appear to be a mapping at the syntax level. After all, at this level the information is serialized in some well-defined, externally readable format that can be processed using standard tools. The mapping might take the form of an XSLT [Cla99] transform that would take a document in one format and, by moving pieces around, translating tags and doing other minor alterations, would eventually output a document in the other format.

The analogy to natural language translation would be a beginning student of a foreign language attempting to translate a sentence with the help of a dictionary, but no knowledge of the grammar. For example, the French sentence "*La chemise est bleue.*" would map word-for-word to "*The shirt is blue.*". However, in a direct mapping of a more complicated sentence like "*On s'ennuie de toi.*" to "*We ourselves bore of you.*" it is nearly impossible to select the correct translation of "*ennuie*" without looking at the context. The purported translation is made up of English words but makes no sense to an English speaker.

Since most synthetic information languages are simpler than natural ones, this approach can work in the computer world, and is especially popular for mapping between various XML vocabularies. It works well when the information structure (in the object layer) is close to the natural structure of the chosen syntax, or when the information structures being mapped are already similar [Vli01].

However, it becomes impractical to use a syntax mapping when the serialization format becomes more complex. Since the transformation is not constrained by the structures of the object layers involved, it can be difficult to produce a document that will deserialize into a valid instance of the model. (Imagine trying to write a compiler that does not create an abstract syntax tree!) Efforts to surmount this problem usually result in the transforming application deserializing progressively larger parts of the document into an information model, eventually resulting in what is effectively an object-to-object mapping. (See Section 5.3.4 for an example.)

In practice, syntax-to-syntax mapping can be used to quickly integrate models with simple syntax that corresponds directly to their semantics, but it is impractical when integration requires structural changes to the model above the level of a simple graph.

2.4.2. Object-to-Object Mapping

Since a syntax-level mapping can quickly get out of hand, an object-level mapping that raises the level of abstraction is the next logical avenue to explore. In this approach, the object-level features of one metamodel are mapped directly into the equivalent features of the other metamodel. The mapping can be prepared in advance and later automatically applied to any number of models.

To prepare an object-level mapping from French to English, we would first consider all possible grammatical forms in French; each would then be mapped to the equivalent expression in English. This would be a monumental task due to the large quantity of grammatical forms and the many exceptions fostered by natural language. In addition, since not all French expressions have an exact equivalent in English, the mapping must be prepared on a best-effort basis, acknowledging that some finer details will be lost. For example, French has many more verb tenses than English, and while they are somewhat interchangeable each gives a slightly different flavour to a statement. Translating a statement

back into French is unlikely to recover the original sentence. As most machine translators work at this object level, it can be very entertaining to translate a sentence back and forth between two languages, watching its meaning drift with each mapping.

Though synthetic languages are simpler, establishing a good mapping is also often a problem. The various metamodels use different fundamental semantic features as part of their object layer. For example, some have a simple source and target for each binary relationship, while others require the roles played by each member to be specified. Some metamodels have no direct support for n -ary relationships (they are modeled in the semantic layer), while others have no standard way of expressing order. There is no agreement whatsoever on more advanced features such as reification and scoping.

Thus, when building a mapping, it is rare to find an exact equivalent for each feature. Richer features that are forced into simpler forms will have to lose some of the data they carry, distorting the information and making roundtrips impossible. Basic features being integrated into a richer metamodel need additional information to be synthesized, which cannot be done without supplementary per-model instructions. Some features do not have even a rough match and must be discarded altogether or lifted to the semantic layer.

2.4.3. Object-to-Semantic Lift

When features cannot be mapped directly, and the information they encode must be retained, they must be described indirectly using the object-level features of the target metamodel. This effectively lifts some object-level features of the source into the semantic layer of the target metamodel. All information can then be preserved, so the lift enables roundtrips. However, tools written for the target metamodel will be unable to “understand” the meaning of the additional information introduced and will only be able to deal with it in a generic manner, since the semantics of the extra features were not programmed into the applications.

The logical limit of this approach is to lift the whole object layer of a metamodel when mapping, using the target metamodel's objects as a structural mechanism stripped of its semantics. This effectively pushes the target object layer into the syntactic layer and has been referred to as "modeling the model" [Moo01]. This approach is often used when integrating a much more complex metamodel into a simpler one, since the number of features that can be mapped directly is likely to be small anyway.

In our analogy, a semantic lift might result in "*La chemise est bleue.*" being mapped to "*A French sentence whose subject is 'the shirt', whose verb is 'to be' in the present tense, and whose predicate adjective is 'blue'.*". This kind of mapping works for all French sentences and the result is a correct English sentence that retains all the information of its French counterpart. It is not, however, a translation: the resulting statement describes the original sentence instead of restating its claims in English. While the meaning can be recovered, it requires a knowledge of the ontology of French grammar.

A corresponding semantic web example is the proposal of a mapping of the complex Topic Maps metamodel to the simpler (but more popular) RDF metamodel [LD01]. The mapping first decomposes Topic Maps object-level features into a syntactic graph (following the [NB01] proposal), then expresses this graph using some of RDF's object-level features, namely identity, binary associations and reification. The mapping is unidirectional, complete and reversible. It is a lift because it introduces an ontology (in the semantic layer) that is necessary to interpret even the basic features of translated models.

Lifts are thus a tradeoff. They allow an automated mapping from one metamodel to another with no loss of information, and enable many tools of the target metamodel to be used (e.g., databases, query engines). However, any manipulation of the translated model using the target metamodel's tools effectively occurs at the syntax level with the concomitant fragility identified in Section 2.4.1,

unless the semantic layer supports imposing constraints on the model to mitigate the issue.

More importantly, though, lifts are inimical to model integration, since a feature expressed directly in the target metamodel will differ from the same feature lifted from the source metamodel. For instance, in the Topic Map to RDF lift discussed above, both metamodels directly support binary associations. In RDF, a binary association is represented as a labelled directed arc between its two members. When the same binary association is expressed in a Topic Map, then lifted into RDF using [LD01]'s scheme, it is represented with no less than 14 arcs between 9 nodes,¹⁰ and those 14 arcs do not even include the single arc of the natural representation. This is a serious problem, since one of the basic tenets of information integration is that there must be only one way to represent each piece of information.

2.4.4. Semantic-to-Semantic Mapping

A mapping can also be carried out at the semantic level by interpreting the information represented in one model then re-encoding it using a different metamodel. This is the task performed by a translator who reads a French document and writes an English one that delivers the same information, though the forms of the sentences may be completely different. This procedure has to be carried out manually by a human with a good understanding of the domain being modeled and of the structure of both metamodels involved. It is the most accurate of all approaches, since the meaning of the model is what humans actually care about. It can be usefully applied to small, high-level ontologies, but does not scale to large models.

¹⁰ If this seems excessive, remember that the Topic Map metamodel is richer and a binary association carries more information than in the RDF metamodel.

2.4.5. Mapping Summary

Mapping is a necessary part of any attempt at integrating information. Neither syntactic nor semantic mapping is practical in the context of the semantic web; the first is too fragile and the second does not scale. The only option left is an object-level mapping that does not perform any lifting. Specifying mappings individually between every pair of metamodels is a futile effort: it results in the Stovepipe System anti pattern [BM+98b], where the number of mappings grows as the square of the number of metamodels. The best solution is to have a single, central metamodel into which all the others can be integrated, with one mapping per foreign metamodel. To avoid losing information, the target metamodel must be richer than all the source metamodels, so that good equivalents can be found for all primitive elements.

The following chapter introduces a new metamodel that satisfies these constraints, and Chapter 4 gives information-preserving mappings from other popular semantic web metamodels.

3. The Braque Metamodel

This chapter describes the Braque¹¹ information metamodel, strongly inspired by the principles and ideas put forth in [Gri82]. We begin by restating the design goals and their implications, and proceed to build up the metamodel from primitive elements to logical constraints. Along with the metamodel, we introduce a visual graph-based notation for representing its instances. This notation is loosely based on UML [OMG01], but was modified to better fit an unstratified metamodel where classes and their instances are likely to appear in the same diagram. ([Sch02] is a draft for a similar notation for the OWL [DC+02] ontology that stays within the UML framework.)

This chapter is written in a tutorial style, introducing features gradually and providing many examples. For a concise reference to the metamodel, refer to Appendix A. Some of the longer and more abstruse discussions of open research questions raised in this chapter can be found in Appendix C.

3.1. Goals and Principles

The primary goal for the Braque metamodel is that it be sufficiently flexible to be able to represent any kind of semi-structured data that might be encountered on the semantic web. It must be able to integrate information from all kinds of sources without resorting to a lift and with no loss of meaning, since it seems unlikely (and perhaps undesirable) that everyone will agree on a single metamodel for all data. The metamodel mappings must be complete and must not require changes to the original metamodels. Flexibility will be evaluated by trying to integrate some other metamodels that have thus far resisted unification (Chapter 4).

In order to be flexible, the metamodel cannot force a strict type system, since this would prevent data integration from untyped metamodels. The metamodel

¹¹ See Appendix C.1 for background information on the choice of name for this project.

should also be as simple as possible. A small number of primitive constructs and the ability to combine them in many ways give it the best chance to be able to adapt to any kind of metamodel we might run into. For example, drawing an analogy to programming languages' data models, there should be no distinction made between primitive values and objects (as in C++ and Java). Rather, all information elements should be treated the same way so as to maximize flexibility (as in Smalltalk). The usual trade-off is a loss of efficiency, which I consider acceptable.

Another feature critical to flexibility is the ability to reference anything in the model. Once again, by analogy to programming languages, we need powerful reflection facilities such as Lisp's ability to reference the program itself as a list, or Scheme's continuations. This leaves the metamodel open to manipulation from within the models themselves. An important aspect of this feature is that the metamodel must be closed under queries: the results of any query will "look" the same as any other part of the model, and can be referenced directly.

Since the new metamodel is meant to be used to integrate data from the semantic web, it must deal with issues of trust. At the very least, it must be easy to identify the provenance of each piece of information, and to filter out information that is not deemed trustworthy. The previous two principles dictate that this should not be a special-purpose mechanism (simplicity) and that it can be achieved by referencing all the data obtained from a source (universal references).

The metamodel is also not meant to be locked in an ivory tower: it should be comprehensible to developers, and allow clear visualizations for end-users. This eliminates the very expressive and powerful approaches based on formal logic statements. The comprehensibility goal will be evaluated by qualitatively comparing the elegance of Braque to other metamodels, and by verifying that the programming interface can be expressed nicely in an object-oriented programming language. The visualization goal will not be evaluated in this thesis.

While it must be possible to build an implementation of the metamodel, its complexity and efficiency are not concerns at this stage. Powerful metamodel features are worth a one-time extra developer effort during the implementation, and optimization should only be performed once critical sections have been identified.

Ease of serialization is not a goal either. If data needs to be exchanged, it can be exported in any of the multitude of other standard formats. Manual data manipulation will be done with the assistance of an application and not directly through a text file. Thus, no serialization format (XML or otherwise) is specified in this thesis.

3.2. Primitives

We first define the truly primitive features of the metamodel—its object layer in terms of the IMI Reference Model. While the reference model considers basic typing and reification to be in the object layer, in this metamodel they are built in the semantic layer using the primitive features, and hence presented in Section 3.3.

3.2.1. Atoms

The first thing we put into the metamodel are atoms. Atoms, also known as urelements¹² or individuals, have no internal structure and reify any thing or concept that we cannot or do not wish to further decompose. Examples include literal values such as Booleans, numbers, characters, dates, etc. We also use atoms to stand for things in the world outside our system, such as books, people, web sites, files, etc. The only difference between the two kinds of atoms is that some may stand for things that have a representation in the programming lan-

¹² “Ur” is a German prefix which is difficult to translate literally, but has a meaning close to “primeval”. In “pure” set theory, all elements are sets and there are no urelements. Often, the axioms of set theory are modified to allow the presence of urelements for ease in representing something. [Wei02]

guage we happen to be using, while others may not; this distinction is mostly irrelevant to the metamodel.

Figure 3-1 shows a few atoms. The first is an anonymous “plain” atom that reifies some object with no direct representation within our system. A model will usually contain a large number of these, each reifying a different object, but the only way to apprehend the referent is by examining the relationships into which it enters. The other atoms reify literals for which some sort of native representation exists. It is usually safe to assume that all systems provide a native representation of strings, Booleans and numbers, so we will make free use of literals of these types throughout the rest of this document.



Figure 3-1. Representation of plain and literal atoms

Each atom has its own unique identity, independent of any value it might be reifying. However, literals are tidy, so there is precisely one atom for each possible value of a literal of a given type. Thus there is only one atom that represents the string “42”, and a separate unique atom that represents the number 42.

3.2.2. Hypersets

The other kind of primitive element is the hyperset. A hyperset that contains other elements reifies a real-world association between the items reified by its members. Structurally, a hyperset is a set, and can contain any mixture of atoms and hypersets. Hypersets differ from classical sets in that any given set is allowed to contain itself. In axiomatic set theory, this is achieved by replacing the Axiom of Foundation with the Anti-Foundation Axiom [Acz88], so hypersets are often called non-well-founded sets.

Each hyperset has its own unique identity, independent of any commonalities in structure with other hypersets. (To determine structural equality, one must resort to the bisimulation algorithm defined in [Acz88], since the usual extensional

definition does not work for non-well-founded sets.) The hypersets' identity feature and ability to hold references to other hypersets combine to give the abilities ascribed to "reification" in the IMI Reference Model, though a more ambitious mechanism is introduced in Section 3.3.7.

Figure 3-2 gives an example of a small model built using atoms and sets. Figure 3-2a employs a simple nesting notation. Sets are drawn as large ellipses, atoms as filled dots, and membership is indicated through containment. The literal atoms' labels merely indicate the value to which each atom corresponds in our interpretation; they are not part of the model. Instead, each atom that has a representation in the underlying system would be mapped to it in an implementation-dependent way.

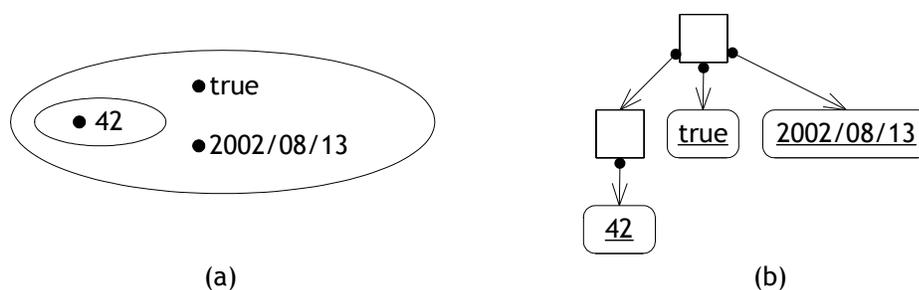


Figure 3-2. Two representations of sets

Figure 3-2b shows the same model as Figure 3-2a, but uses a custom-made directed graph notation loosely based on the one presented in [Acz88]. Sets are drawn as empty boxes and literal atoms as boxes with their value underlined. Arrows with a filled dot at their origin point from sets to their members.

You can tell that both sets in Figure 3-2 are classical because the graph in Figure 3-2b is acyclic. Figure 3-3 shows an example of a recursive hyperset that generates an infinite expansion. It is not practical to represent such models using nested diagrams, as illustrated by the Escher print in Figure 3-4. The artist rendered a picture that contains itself, but had to leave a blank spot in the center since it is impossible to directly represent this kind of embedding. The same

problem occurs when trying to draw a self-nested set, so the rest of this thesis only uses the directed graph representation.¹³

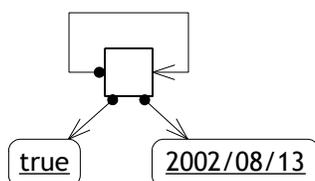


Figure 3-3. A recursive hyperset



Figure 3-4. M.C. Escher's "Print Gallery" (1956)
©2002 Cordon Art - Baarn - Holland.
All rights reserved. Used by permission.

With only two kinds of primitive elements, the metamodel is very simple. It could be pared down further by defining atoms to be empty hypersets. There is no benefit in such a redefinition, though, since it does not make the metamodel any more expressive and it may be confusing to model atomic values as empty sets. The metamodel also allows for literal nests, if it happens that the underlying system can directly represent the object that the nest reifies; these are rarely used.

3.2.3. Identity

While it was mentioned in passing above, it is useful to expand a little on the notion of element identity. How, exactly, is each element identified? Since we are

¹³ Section 5.1.1 explains why this is only a representation and how the metamodel differs from a traditional directed graph.

dealing with the Semantic Web, URIs appear to be a good first candidate for the job. They are uniform, and the social conventions surrounding them should ensure a unique authority to determine the referent of each one. On the surface, it seems that using URIs as the primary identifiers for our elements (like in RDF) is a good idea: each object would be internally coupled with a URI, and each URI would be tied to at most one element.¹⁴

On reflection, the idea's appeal fades. As explained in Section 2.1.3, there is no widespread agreement on what each URI identifies. When interpretations collide, it is important to be able to reason about what different people believe each URI identifies. This is impossible if the URI is hardwired into the element.

This situation bears some similarity to the debate on natural versus surrogate primary keys in the relational database community. A natural primary key is formed using an existing subset of columns in a table such that each row is assumed to have a unique unchanging tuple "key", dictated by the business rules governing the data. The widely noted problem, of course, is that business rules change and humans are not perfect, so natural primary keys end up neither fixed nor unique, upsetting all other tables that use them as foreign keys. The widely adopted solution is to employ a surrogate primary key by adding a new column to each table that will hold arbitrary unique and unchanging values unrelated to the meaning of each record. A good application will hide the values of these surrogate keys from users, expressing them as relationships between data records.

The answer for our metamodel follows this pattern. Each idea has an intrinsic, unique, unchangeable identity that is not externalized in any way; this encapsulation can be likened to using opaque references instead of address pointers in a programming language. Other than identity creation and deletion, which coincide with the creation and deletion of ideas, only two other identity operations

¹⁴ Strictly speaking, this is not true for RDF since the standard also allows "blank nodes". These objects are existentially qualified and have no well-known identifier.

are permitted. First, two identities can be compared for equality, to see if two references to ideas really refer to the same one. Second, an identity can be replaced by another (existing) one; atomically, all references to the idea with the first identity are replaced with references to the second, and the first idea is deleted. This operation is similar to the “become:” primitive in some dialects of Smalltalk.¹⁵ [GR89]

3.2.4. Ordering and Duplicates

While the metamodel as presented above is formally complete, the models are meant for human consumption and people often like to have data elements consistently ordered. It is possible to encode order directly with sets, so adding it as a primitive will not increase the expressive power of the metamodel but it will make ordering elements far more convenient and efficient.

The most common way to order elements is to impose a total order on the set, making a chain. This is not enough, though. When integrating data obtained from different sources, it is likely that we will have to merge ordered sets together. Since we expect the data to be semistructured, it is possible that the sets’ members will not be comparable to each other. We must thus include partially ordered sets in our metamodel to satisfy the goal of smooth integration.

It is also useful to relax the set restriction and allow duplicates into the collections. In this way we obtain lists / n -tuples (when totally ordered) and bags / multisets (when unordered). The former are especially useful, since pairs can be interpreted as directed arcs and used to attach properties to elements (see Section 3.2.5).

¹⁵ In the original Smalltalk-80, “become:” swaps the two identities. In some later versions, the original identity is destroyed as part of the operation, leaving only the new one.

The graph representation has only basic facilities for indicating primitive order, since complex orderings are usually inferred from semantic features anyway.¹⁶ Figure 3-5 illustrates a nest that contains two members: the literal *true* in the first position, and the date *2002/08/13* in the second. Order is shown with increasing numbers of tick marks, though the technique is only practical up to three or four members.

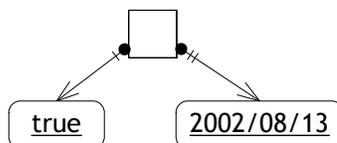


Figure 3-5. Ordered members

Table 3-1 shows some of the kinds of collections possible in the metamodel, organized along the axes of ordering and duplication and stripped of the “hyper” prefix. We will use the term *hypernest* (or *nest* for short) when referring to a collection that is part of a model but whose exact kind is irrelevant.

	Unordered	Partially Ordered	Totally Ordered
No Duplicates	set	poset	chain
Duplicates Allowed	bag	pomset	list / <i>n</i> -tuple

Table 3-1. Kinds of nests

Partially ordered multisets (pomsets) are the most general type in the table above; all the other types can be considered as restrictions of pomsets [GM95]. We can thus simplify the metamodel to one of atoms and hyper-pomsets without losing the familiar collection types familiar to software developers. This list of collection types above is not exhaustive, since other kinds of pomset restrictions could be considered (e.g., multi-linear). For this reason, and because the exact restriction type is usually relevant only to the implementation, the graph representation of the models does not directly indicate the kind of each nest.

¹⁶ For example, a partial order is most easily indicated with an acyclic directed graph. This graph, built using totally ordered pairs, would then induce a partial order on the collection of elements so linked.

3.2.5. Nest Size

There is one last “primitive feature” that should be explicitly stated: the meta-model allows nests to have an infinite number of members. That is, nests can be of any cardinality. This is hardly surprising from a mathematical point of view, but bears affirming in a computer science context. The need for infinite extent will become clear when the naïve typing system is introduced in Section 3.3.1.

Some examples of nests with infinite extent are the sets of numbers, moments in time, and positions in space. Notice that many of the infinite sets are dense, or even uncountable. They have a natural total ordering of their members, yet there exists no sequence that would enumerate them in the correct order. These infinite chains cannot be ordered using ordinal properties or linked lists, the only techniques considered in the metamodel survey [MD00].

3.2.6. Notational Sugar

All the metamodel primitives have now been introduced. The graph representation, though unable to express every possible model, is sufficient for our needs here. Nevertheless, manually expressing some common higher-level features in terms of nest membership quickly gets laborious, so it is worth introducing a few notational shortcuts. Each shortcut is directly equivalent to a structure built out of the primitives introduced above, so this section does not actually introduce any new features.

Thus far, all the nests have been anonymous, their identity determined by the graphic’s location in the picture. When talking about larger graphs, it is useful to be able to refer to specific nests or atoms in the text, which is inconvenient for anonymous rectangles. Obviously, the solution is to label them, but note that this label is only an externalization of the element’s innate identity and is not itself part of the model. For example, in Figure 3-6, nest *A* is structurally the same as both anonymous nests (though each maintains its own separate identity), while nest *B* holds non-literal atom *C* as a member.

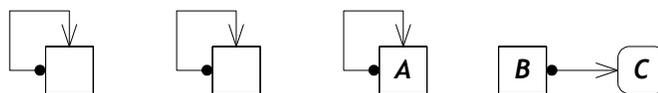


Figure 3-6. Labelling elements to externalize their identity

With labels, we can also decouple the identity of an element from its visual representation. One approach would be to say that any two components sharing a label represent the same element. However, this can lead to inadvertent label collisions unless disjoint namespaces are strictly enforced. Since this is an informal notation, we instead differentiate between labels that name a new element being introduced versus labels that refer to a previously declared element. This prevents accidental aliasing (since each labelled element is unique) at the expense of occasional referential ambiguity, tempered by the ability to scope the labels.

Figure 3-7a declares the nest A and two distinct atoms both labelled B . Figure 3-7b shows an ambiguous reference to some atom called B . Figure 3-7c shows a scoped reference to the atom B contained in the nest A . References can be scoped to any nesting level.

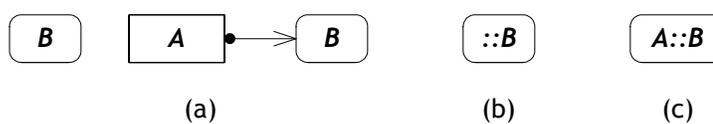


Figure 3-7. References and scoping

Nest membership, being the fundamental relationship, sees extensive use. It is useful to have a few different ways of expressing it to make a diagram clearer. Figure 3-8 shows four equivalent ways of expressing two membership relationships. In Figure 3-8b you can read “: A ” as “in A ” (or, as explained in Section 3.3.1, “of type A ”). There can be multiple such annotations, one per line, and they are all implicitly references so the nest must be declared elsewhere. In Figure 3-8c, visual containment indicates membership, and the contained elements can be declared *in situ* or referenced as explained above. This notation is

especially useful when B is being used as a namespace. Figure 3-8d shows a combination of both alternative notations.

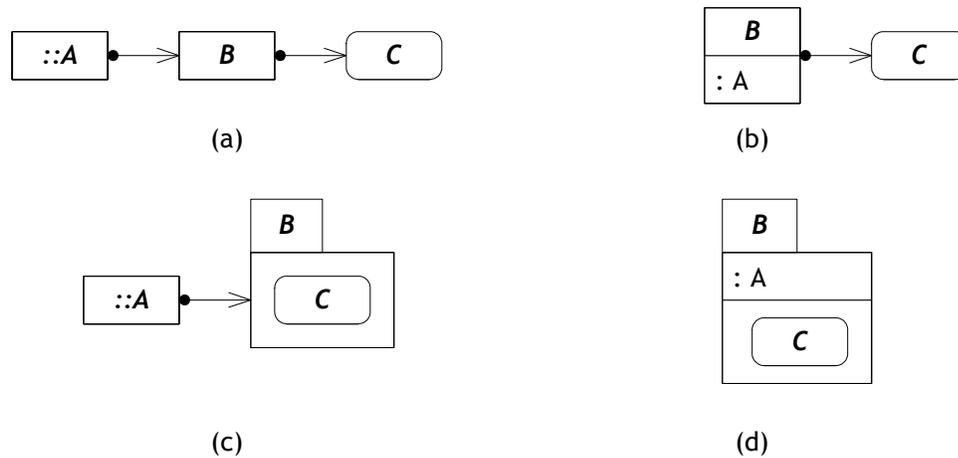


Figure 3-8. Equivalent representations of membership

Finally, since models make extensive use of binary relationships, having a concise way of expressing them would greatly improve readability. For this purpose, we use a simple arrow from the origin to the target; the two parts of Figure 3-9 are equivalent. When using the shorthand notation in Figure 3-9a there is no way to refer to the relator nest identified as P in Figure 3-9b.

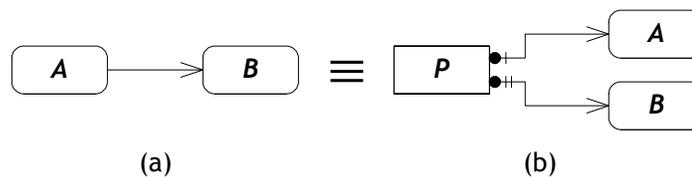


Figure 3-9. Binary relationship shorthand

More graphical abbreviations will be introduced as we expand the metamodel's semantic layer in the following section.

3.3. Naïve Upper Ontology

At this point, the metamodel is still missing some features that many would consider basic—typing and identification, for example. These features are not primitive in the metamodel, but rather built in the semantic layer using the fundamentals introduced above. The construction is naïve, relying on intuitive definitions

of the necessary concepts rather than fancy technical approaches. Thanks to this lack of extensive technical constraints, other upper ontologies can easily be embedded in the naïve upper ontology.

Those advantages notwithstanding, it is important to realize that this naïve upper ontology (NUO) is not a part of the metamodel. It could be replaced with another upper ontology, for example a stratified one, or one specific to the models under consideration. It could even be scrapped completely if typeless, nameless models are sufficient for the task at hand. However, for the purposes of integration, a light, naïve upper ontology is a desirable asset.

3.3.1. Types

While the metamodel does not enforce strict typing, it is useful to look at how classes might be represented. The most intuitive approach is to consider a class as the set of its instances. The instances can be added to the class set manually, which would directly correspond to an extensional definition of the type. The contents of the type set can also be the result of a query, perhaps based on some properties of the class set, corresponding to an intensional definition. The definition could even be mixed, such as for a concrete superclass that automatically contains all the instances of its subclasses, but may also have explicit instances of its own.

Figure 3-10 shows an example of the class of *Booleans*. The class contains two literal atomic instances, true and false.

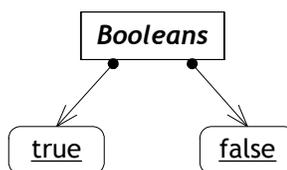


Figure 3-10. Booleans class

In a traditional class-based metamodel, and in accordance with intuition, all objects should be instances of at least one class. What is the class of the *Booleans*

nest? Since *Booleans* is a class, we need to introduce a class of classes, which we will call *Classifiers* (Figure 3-11).

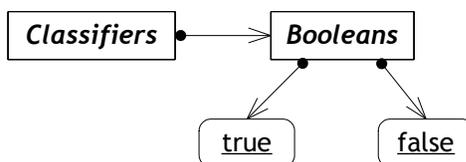


Figure 3-11. Booleans is a classifier

We are now left to ponder the type of the *Classifiers* nest. Let us informally define the *Classifiers* class as “all nests that impart some common characteristics to their members, which are considered their instances”. It is then clear that *Classifiers* is a classifier itself, since it contains nests whose common characteristic is that they are all classes that are considered instances of the *Classifiers* class. Thus, *Classifiers* is a member of itself (Figure 3-12), providing one reason why non-well-founded sets are needed.

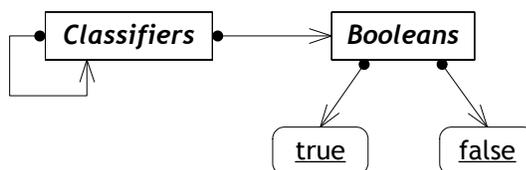


Figure 3-12. Classifiers is an instance of itself

This approach to typing gives the metamodel an unstratified (non-fixed layer) architecture. The advantage is that types and instances are all part of the same model and can be treated uniformly, which is very important for semistructured data [ASB99]. The purported disadvantages were mentioned in Section 2.2.2. However, those objections are irrelevant in view of the project’s stated goals: since there are other metamodels that use a non-fixed layer architecture (for example RDF [LS99]), Braque must support it too to be able to integrate them.

3.3.2. Relations

With a basic type system available, we can now define relations. We will use the mathematical definition of a relation: it is a set of n -tuples, usually of the same

ordinality [Wei02] (see Section 3.4.2 for details on expressing this constraint). For now, we will only define binary relations, since those are the most commonly used. Figure 3-13 show the *Binary Relations* class and a sample instance of it, the relation *Length* that establishes the correspondence between a few strings and their length.

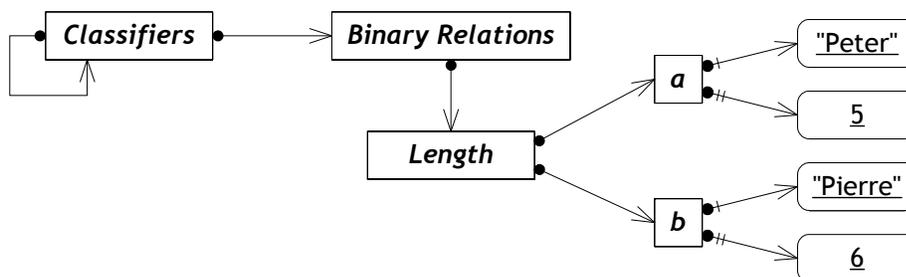


Figure 3-13. Example of a binary relation

This representation is a little unwieldy, but we cannot replace the two relator nests *a* and *b* with the shorthand arrow notation since we need to refer to them to show their membership in the *Length* relation. As this pattern is repeated quite often, it is worth introducing a new shortcut notation for typed binary links (Figure 3-14).

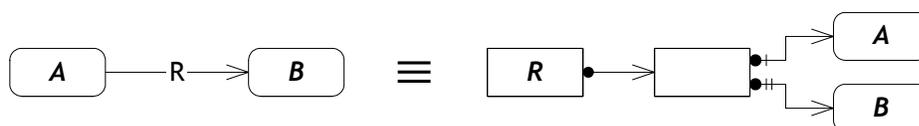


Figure 3-14. Typed relationship shorthand

We can now redraw Figure 3-13 as Figure 3-15, preserving the original meaning (though the relators are now unlabeled). Note that using *Length* as the type of a relator does *not* automatically imply that *Length* is an instance of *Binary Relations*. This relationship must still be asserted explicitly, since the relation might actually be an instance of some specific subtype of *Binary Relations*.

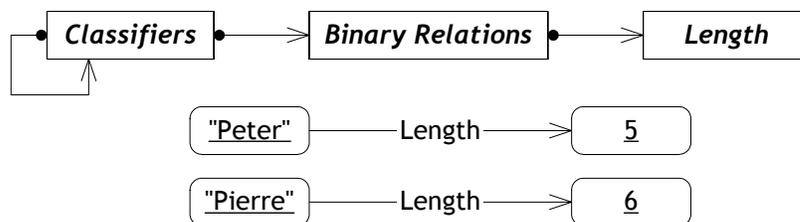


Figure 3-15. Example of binary relationship shorthand

Do not confuse the shorthand link notation with a labelled graph. Each relator is still an individual nest with identity, not just a labelled arc.

3.3.3. Membership Reification

In a model, reification is the action of making some implicit object “real” within the model, and thus gaining the ability to refer to it. Most commonly used is reification of links (statements), for example to make further statements about their origin. The Braque metamodel naturally caters for this common case since links are just nests (relators), and can already be referenced directly in other nests.

Reification of membership is more rarely used. The intention is to reify the component objects of a link to make statements about them. In Braque, this is equivalent to reifying the membership of each element. For n -tuples, you can think of it as reifying the endpoints of the binary link.

The mechanism is as follows. Each model that requires reification includes a special *Member* relation. It contains one container-member pair nest for each such pair present in the model. For bags and lists, each duplicate member gets its own reified membership pair. The *Member* relation is partially ordered: the pairs for each nest match the order of its members, and pairs corresponding to different nests are unrelated.

Note that the *Member* relation is defined over the whole model, and is itself part of the model. Consequently, if a model contains at least one non-empty nest, the *Member* relation will hold infinitely many membership reification pairs. The first pair reifies a member of the non-empty nest, which introduces a nest with two

more members into the relation. The nest's membership in *Member* and its own members are reified, in turn, and produce three more pairs, etc. ad infinitum.

Membership reification is rarely needed, but can be very useful when we need to distinguish between duplicate elements in a nest and when integrating meta-models (see sections 4.2.6 and 4.3.1). It is almost never useful past the first level [New02], but the metamodel must support arbitrary reification to be consistent. The reification is also pre-emptive rather than lazy (on-demand) [New02]; this creates a strong connection between the membership and its representation, and ensures that each model does not create its own reifications of a common concept.

3.3.4. Roles

In mathematics, it is common to identify the meaning of the members of a relationship by their position in the relator. That is, for a *Length* relation, it is understood that the object whose length is being measured is stored at index 1 in each relator, and the length of that object is stored at index 2. These implicit conventions are sufficient for simple relations, but—just like record structures in most programming languages—it is clearer and more flexible to explicitly identify the role played by each member of a relator.

Notice that in general it is not enough to establish a correspondence between member positions in a relation and the roles played by the elements they hold. Not all relations have position-based parameters, or even a fixed arity (for an example, see Section 4.3). It is thus important to identify the role played by each member in each individual relator (instance of a relation).

To achieve this, it is not enough to link each member of a relator to its role: the same element might be playing a different role in another relator, and there would be no way to distinguish between them. It is not even enough to use a ternary link between the member, the relator and the role, since the same element might appear in different positions in the same relator, and play a different role

for each position. With a ternary link, there would be no way to correlate the roles played to member positions.

A better solution is to take advantage of membership reification and link each *Member* pair to the role it fulfills. This attaches the role to all the information available about the membership of an element in the relator, including its position. The most elegant way to implement this link is to consider each role as a classifier containing the reified membership elements that fulfill that role. According to this definition, the *Member* relation is our first role classifier representing the role of a generic nest member. All other role classifiers will expand it.

Classes are often closely related to the roles their instances play. For example, “SEng 330 is a course” and “the SEng 330 course is taught by the teacher Piotr” are kin statements, though the first one uses “course” as a normal classifier and the second as a role.¹⁷ To relate a classifier to roles its members play we will use the *Enact* relation, linking the role to the classifier. If the definition of the role does not add any extra information to the classifier it enacts, we will use the *Enact default* relation—there should be no more than one default role for any given classifier. A default role is most useful when all members of a class can play only one role in the relationship.

Figure 3-16 demonstrates the role-playing mechanisms by showing two relationships: “the course SEng 330 is taught by the teacher Piotr” (relator *a*) and “SEng 330 requires CSc 115” (relator *b*). In *a*, *SEng 330* plays the default role of a course, since there’s no advantage to expanding it to “the course being taught”. In *b*, it is critical to distinguish between the two roles (both played by courses), since one is the required course, while the other is the requiring one. Using the default course role in this relationship would leave us unable to disambiguate, since the relationship is not ordered.

¹⁷ Some metamodels do not make this fine distinction; see Appendix C.4 for an explanation of the problems that occur when “normal” classifiers are also used as roles.

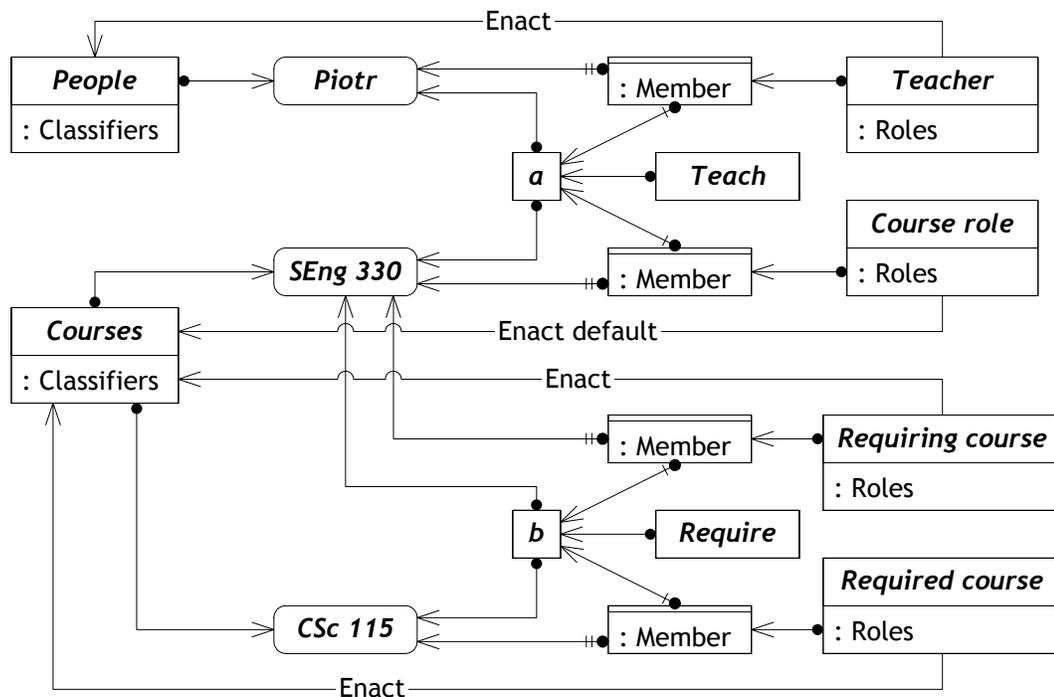


Figure 3-16. Example of relationships tagged with roles

Figure 3-16 is accurate but too complicated to be easily readable. To simplify graphical representation of roles, we introduce two new shorthands. Figure 3-17 shows a quick way to indicate the role played by a containment relationship, which can be used to compact diagrams of n -ary relationships with role-playing elements. Figure 3-18 proposes a further refinement of the notation for unordered binary relationships, where members are distinguished by the roles they play. (This last notation can be combined with the relationship-arrow notation to represent an ordered binary relationship with members that play roles.)

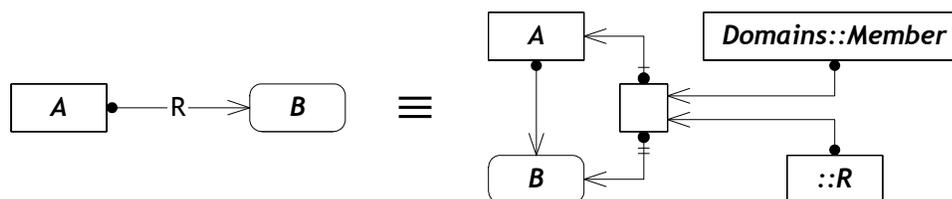


Figure 3-17. Membership role-playing shorthand



Figure 3-18. Unordered binary role-playing relationship shorthand

Using this new notation, and assuming that the classifiers, relations and roles referenced are defined elsewhere, we can transform Figure 3-16 into the much simpler Figure 3-19.



Figure 3-19. Example of role-playing relationships using compact notation

Of course, all the relations in the NUO actually have totally ordered relators, where the meaning of a member is indicated by its position. To integrate this approach with roles, the *Play Role by Index* ternary relation assigns a role to each index of a relation. Since *Play Role by Index* uses indexed parameters as well, we can use it to describe the roles played by the members of its own relators (Figure 3-20). Section 3.4.2 shows how roles played in actual relators are inferred from these statements about relations.

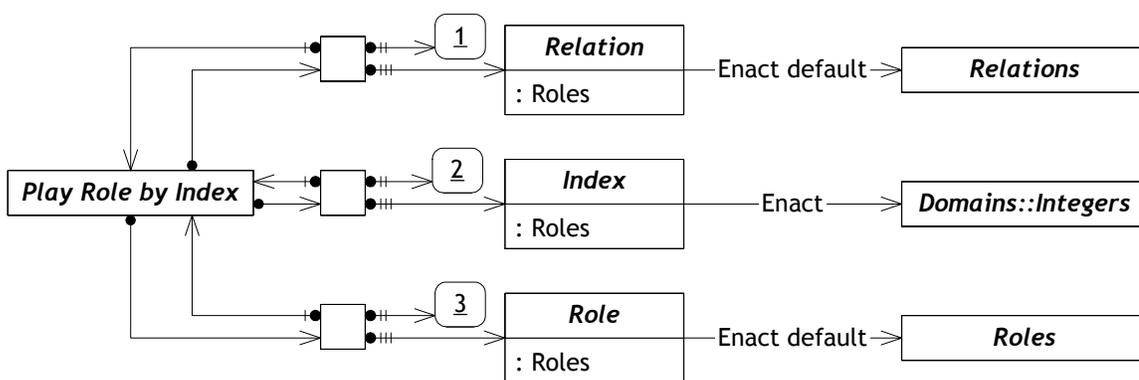


Figure 3-20. Roles played by members of Play Role by Index relators

3.3.5. Subtypes

All metamodels that support types (whether in the object or semantic layer) also support type specialization, so it is worthwhile to add the idea of subtypes to the NUO. A subtype A that specializes a type B may add further characteristics to its instances, but every member of A is also a valid member of B . With our definition of classes, a subclass is merely a subset of the union of its superclasses. We introduce a new binary relation *Extend* that indicates specialization, relating the subclass to the superclass.



Figure 3-21. The Extend relation

We can now use the relation to indicate that *Extend* is actually a specialization of *Expand*, the relation that indicates that one set is the subset of another.¹⁸ We could use the previously introduced relationship shorthand but, since specialization is used so often, we introduce an even more compact abbreviation inspired by UML ([OMG01][Fow00]). All parts of Figure 3-22 are equivalent.

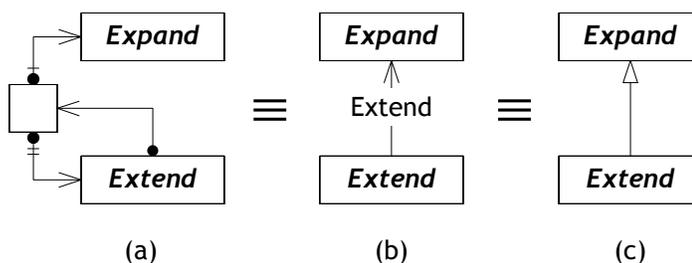


Figure 3-22. Extend relationship example and shorthand

With extension at our disposal, we can now begin building the “top” of the NUO’s inheritance hierarchy. First, we need a top-level classifier that contains all elements in our system; this is useful when writing constraints (see Section 3.4) and queries. Most of the good names are already taken: object, entity, concept, resource, topic, subject, etc. all have existing connotations. To avoid preconcep-

¹⁸ The difference between the two relations is explained in Section 3.4.4. For now, you can consider them to be equivalent.

tions, the class of all elements in the model is called *Ideas*. Every element is implicitly a member, so we do not indicate this membership in the diagrams. Similarly, *Nests* are a subclass of *Ideas*, and every nest in the diagram (sharp-cornered rectangle) is implicitly a member. Finally, *Classifiers* are a subclass of *Nests*, and all of these are instances of *Classifiers*.

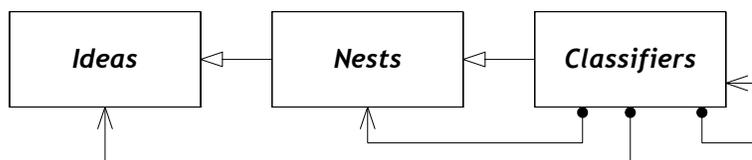


Figure 3-23. Top of the inheritance hierarchy: Ideas, Nests and Classifiers

With this basis in place, we can now attach a hierarchy of relation metaclasses such as *Relations*, *Binary Relations*, etc. Instances of those classes will be relations,¹⁹ which can also be seen as classes of relationships. These relations will contain the tuples that define the individual relationships.

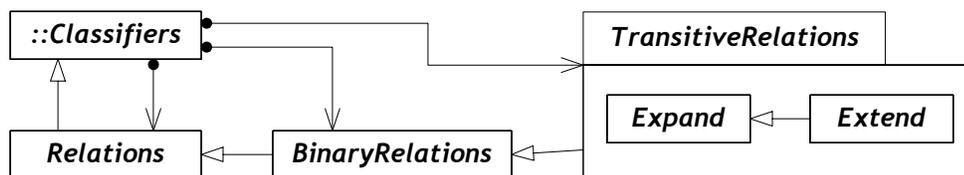


Figure 3-24. Relations inheritance hierarchy

3.3.6. Names

Humans like naming things: having a name makes it easier to refer to the object. The labels in the graphical representation allow us to refer to the nests and atoms in the text but are not part of the model itself. It would thus be useful to introduce a standard intra-model naming mechanism, to simplify queries on and presentation of models.

Two principles guide the naïve design. First, each object may have multiple names and a name can denote multiple objects (for unique identification, see Section 3.3.7). It is thus inappropriate to store a single name within the object itself;

¹⁹ See Section C.1 for a discussion about the use of instances versus subclasses in this case.

the relationship should be made explicit. Second, a name is not the same as its representation. For example, a student's first name might be Peter. If the student goes to a conference in France, though, he would be called Pierre—the French translation of Peter. Intuitively, the student has only one first name, but it may be represented in different ways depending on context. Figure 3-25 shows a model of this situation, alluding to the ontology of scopes introduced as part of the Topic Maps mapping in Section 4.3.

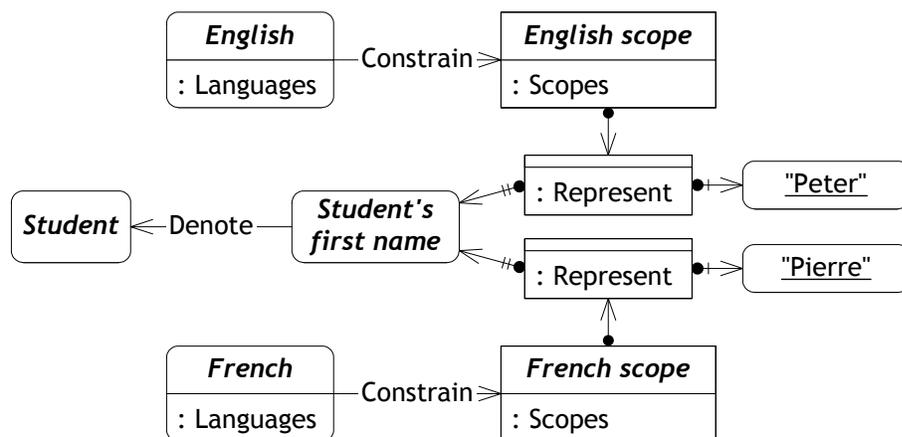


Figure 3-25. A name with scoped representations

Most of the time, we are not worried about multiple representations and do not bother scoping the *Represent* relationships, though we must still use a mediator between the string representation and the idea being named for consistency's sake. Figure 3-26a shows a typical usage of naming. Note the low-key introduction of the *Denote* and *Represent* (non-transitive) binary relations.²⁰ Since naming ideas is a very common activity, a very compact shorthand version is demonstrated in Figure 3-26b. The form is purposely close to the labelled notation since the semantics are very similar. The only difference is that names are recorded within the model, while labels are diagrammatic artefacts.

²⁰ See Section C.3 for further details about these particular relations.

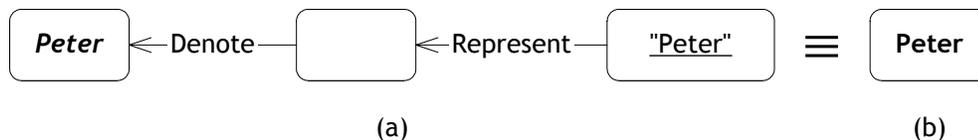


Figure 3-26. Simple naming example and shorthand

Now that names are available to us, we naturally want to name all the standard ideas introduced so far. This is straightforward, though it is interesting to see how the *Denote* relation names itself (Figure 3-27).

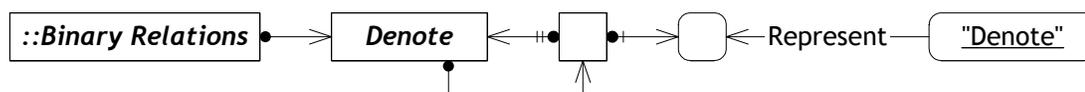


Figure 3-27. Naming the Denote relation

A final interesting example of naming is the situation presented by Lewis Carroll in the excerpt below [Car60]:

"[...] The name of the song is called 'Haddock's Eyes'."
"Oh, that's the name of the song, is it?" Alice said, trying to feel interested.
"No, you don't understand," the Knight said, looking a little vexed. "That's what the name is called. The name really is 'The Aged Aged Man'."
"Then I ought to have said 'That's what the song is called'?" Alice corrected herself.
"No, you oughtn't: that's quite another thing! The song is called 'Ways and Means': but that's only what it's called, you know!"
"Well, what is the song, then?" said Alice, who was by this time completely bewildered.
"I was coming to that," the Knight said. "The song really is 'A-sitting On A Gate': and the tune's my own invention."

Figure 3-28 shows a model that matches this description. Note the strange usage of strings for both the name of the song and the song itself. Indeed, as [Nag56] noted, this last is almost certainly a mistake, but our ontology rises to the challenge and allows us to represent even semantically suspect declarations.

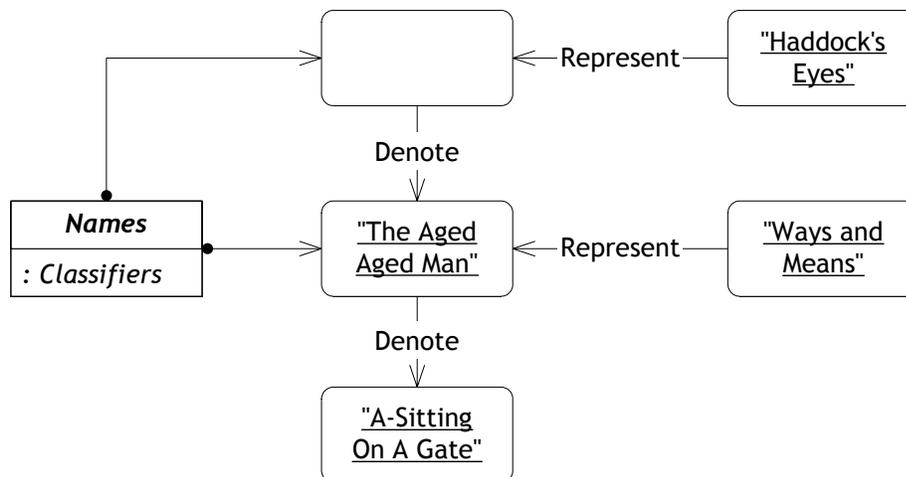


Figure 3-28. A model of the names of “A-Sitting On A Gate”

3.3.7. Identifiers

In Section 3.2.3 we decided to use surrogate keys to identify our ideas; how then do we represent external identifiers, such as URIs? The key idea is that URIs are just names that are *meant* to be independent of context. While we may not entirely trust this, it is useful to indicate that a name is supposed to be unambiguous. To this end, we refine the naming relation hierarchy tacitly introduced in the previous section as shown in Figure 3-29, where each relation is annotated with its intended meaning.

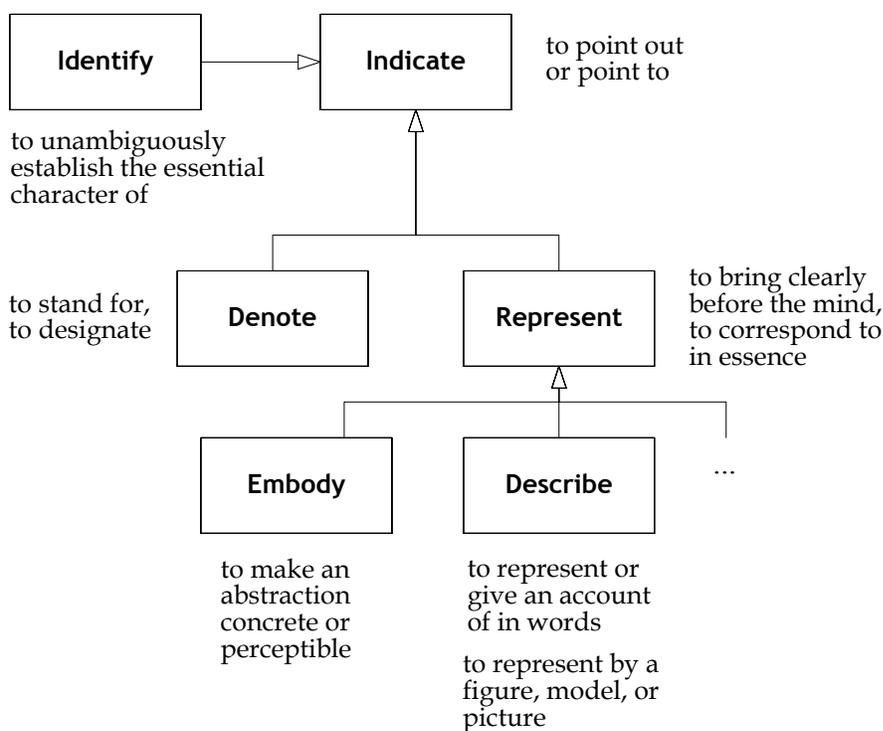


Figure 3-29. A hierarchy of indication relations

Starting out with *Denote* and *Represent*, we create a common supertype *Indicate* since the purpose of both relations is to have one idea point to another. The difference is that *Denote* is meant for symbols (or other things) that are arbitrarily assigned a meaning, whereas *Represent* is for things that indicate through some replication of essential characteristics of the referent. The distinction may not always be clear-cut in practice, so it is acceptable to have one idea *Indicate* another without being either a name (*Denote*) or a representation (*Represent*).

The subtypes of *Represent* show some of the possible ways to represent an object. They are not included in the ontology since there is no limit to the variations and the semantic details do not add useful information to most models.

Finally, the *Identify* relation is a subtype of *Indicate*. It can be used directly, to imply that all items pointed to by the identifier are supposed to be the same one. It can also be used in conjunction with *Denote* and *Represent* to combine their semantics, since both denotations and representations could be sufficiently unambiguous to serve as identifiers. For example, since URIs (should) be unambigu-

ous, and a string represents a unique URI, a relationship between a URI and its referent could be modeled as in Figure 3-30. Note that in this model a URI is considered as an abstract name with a string embodiment.²¹

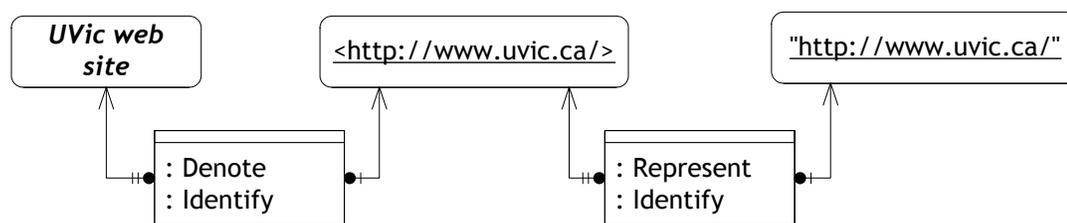


Figure 3-30. Example of identification by URI

While by no means complete, this simple hierarchy of indication is flexible enough to extract some useful common meaning from the various denotation semantics employed by other metamodels.²²

3.4. Inferences and Validation

All metamodels restrict the structure of their models by virtue of the basic definitions of the elements. For example, in our metamodel, it is an axiom that an atom cannot contain any elements. However, it is often useful to impose further constraints on specific models. These validity constraints are the traditional province of schema languages (e.g., XML Schema [TB+01][BM01], RELAX-NG [CM01], RDFS [BG02], TMCL [Pep01]).

Constraints go hand-in-hand with inference, the ability to derive new knowledge from an information base and some rules. Where a constraint checks whether information fits some pattern, inference uses the pattern to create the extra information if necessary. Both constraints and validation can often be expressed using the same logic rules; the only difference is how they are applied. Queries are also closely related to this topic, since they can be construed as constraint

²¹ While technically URIs differ from URI references, which allow for fragments and relative forms, this thesis doesn't apply this distinction and uses "URIs" to cover all these concepts.

²² See Section C.3 for further discussion of these relations and how they apply to the web.

predicates used to select pieces of information from the model, or as inference patterns that create new relationships between existing information.

The Braque metamodel does not specify formal constraint, inference or query techniques. Nonetheless, the graphical notation includes some symbols for hinting at the domain and range of relations and specifying the roles that members of their instances play. Some of the NUO relations also have special constraints that are worth noting, though the metamodel lacks mechanisms to automatically enforce them at this time. The constraints on the metatypes that can participate in an *Extend* relationship are especially complex.

All constraints are expressed in formal logic notation, using only the standard quantification, conjunction, disjunction and membership operators. We add a shorthand notation for indexed relation members since they appear so often in the constraints. Equation 3-1 show the definition for binary relations, higher arities follow the same pattern.

$$R(x, y) \equiv \exists r : (r \in R \wedge r[1] = x \wedge r[2] = y)$$

Equation 3-1. Binary relation formula shorthand

We also use $x[n]$ to refer to the n 'th member of the nest x , with the first member having index 1. If the nest is not totally ordered, then $x[n]$ is satisfied by all elements that can trace a path of immediate precedents of length $n-1$ back to an elements with no precedents. To put it another way, if all maximal totally ordered lists are extracted from nest (such that no element could be inserted between any pair of elements without breaking the total order), then the n 'th element of each list satisfies $x[n]$.

3.4.1. Relation Hints

Thus far, three kinds of arrows were introduced in the graphical notation: membership, relationship and extension. The first two can also be drawn dashed between nests (usually between classifiers) to indicate that these nests' members

are likely to enter into the corresponding relationship.²³ The extremities of the relationship hint arrow can optionally be adorned with the names of the roles played by members that appear at the given end. These are translated into instances of *Play Role by Index* as shown in Appendix A. If no role is specified at an extremity, it's safe to assume that members play the default role for the class at that end. It is also possible to specify multiplicity restrictions, à la UML.

Figure 3-31 shows an example that hints graphically at the domain and range constraints on some of the relations introduced above. The dashed membership arrow shows that the members of *Nests* will contain members of *Ideas* (in other words, nests contain ideas). The dashed *Member* arrow duplicates this information, since the *Member* relation mirrors actual nest membership. The dashed *Represent* arrow shows that any idea can represent any other. These are the widest possible domain/range constraints and the default for any relation, and would not normally be drawn explicitly. However, in this figure, we've also added explicit roles for the relation. The dashed *Expand* arrow also shows roles, but limits both the domain and range of the *Expand* relation to nests.

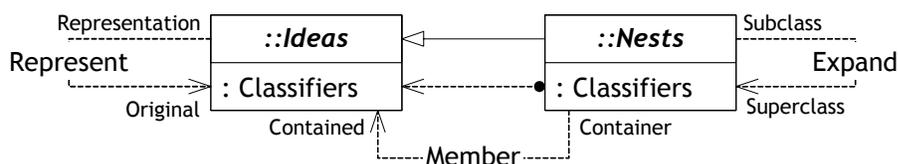


Figure 3-31. Example of relation hints

The relation hints are similar to UML associations: they are drawn between classifiers and refer to an entire relation class of underlying relators. They differ in that the constraints are not strict, and the relation itself still needs to be defined separately. Still, the hints are informative and easy to read, so we will make some use of them in the following chapter when describing more complex ontologies.

²³ There is no need to provide a special hint notation for the extension arrow, since its domain and range are already fixed (see Section 3.4.4).

3.4.2. Relation Constraints

This section examines some of the constraints applicable to the relations introduced in the NUO.

First, we can state that binary relations consist of pairs (Equation 3-2). In the constraint, we need to use a classifier and a relation that have not yet been defined. *Ordered Nests* is a refinement of *Nests* that contains all nests that are totally ordered, and the *Size* binary relation associates each nest to the quantity of its members (which may be infinite).

$$\begin{aligned} &\forall f : f \in \text{Binary Relations} \\ &\Rightarrow \forall x : \left(\begin{array}{l} x \in f \\ \Rightarrow \text{Size}(x,2) \wedge x \in \text{Ordered Nests} \end{array} \right) \end{aligned}$$

Equation 3-2. Binary relations consist of pairs

We follow up by stating the mathematical rule for transitive relations (Equation 3-3).

$$\begin{aligned} &\forall r, x, y, z : \left(\begin{array}{l} r \in \text{Transitive Relations} \\ \wedge r(x, y) \wedge r(y, z) \end{array} \right) \\ &\Rightarrow (r(x, z)) \end{aligned}$$

Equation 3-3. Transitive relation rule

The rule for reflexive relations (Equation 3-4) requires us to define a new relation *Domain* that associates a reflexive binary relation with the nest of ideas that can participate in the relation. To avoid the question of what happens if a relation has multiple domains, we further define *Domain* to be an instance of *Functions* whose members are allowed only one value for any given key (Equation 3-5).

$$\begin{aligned} &\forall c, r, x : \left(\begin{array}{l} r \in \text{Reflexive Relations} \\ \wedge \text{Domain}(r, c) \wedge x \in c \end{array} \right) \\ &\Rightarrow (r(x, x)) \end{aligned}$$

Equation 3-4. Reflexive relations rule

$$\forall r, x, y, z : \left(\begin{array}{l} r \in \text{Functions} \\ \wedge r(x, y) \wedge r(x, z) \end{array} \right) \\ \Rightarrow (y = z)$$

Equation 3-5. Functions have only one value for each key

Identify is another example of a function, since identifiers are supposed to be global and denote at most one idea. As this assumption may not always turn out to be right, it would be wise to apply this rule selectively.

Completing our trio of common relation subtypes, we define symmetric relations. Instead of doing so directly, we first introduce the *Invert* relation, whose instances link a binary relation to its inverse, switching the order of members of each pair (Equation 3-6). A symmetric relation is then one that inverts itself (Equation 3-7). Naturally, *Invert* is itself symmetric.

$$\forall r, s : (\text{Invert}(r, s)) \\ \Leftrightarrow \left(\begin{array}{l} \forall x, y : (r(x, y)) \\ \Rightarrow (s(y, x)) \end{array} \right)$$

Equation 3-6. Inverse of a binary relation

$$\forall r : (r \in \text{Symmetric Relations}) \\ \Leftrightarrow (\text{Invert}(r, r))$$

Equation 3-7. Symmetric relations rule

As promised earlier, Equation 3-8 shows how *Play Role by Index* statements connect indexed members to roles and vice-versa.

$$\forall r, i, o : \left(\begin{array}{l} \text{Play Role by Index}(r, i, o) \\ \Rightarrow \left(\begin{array}{l} \forall t, s : (t \in r \wedge s = \text{Member}[i] \wedge s[1] = t) \\ \Leftrightarrow (s \in o) \end{array} \right) \end{array} \right)$$

Equation 3-8. Relating roles and indices

This leaves only a few loose ends to tie up. Equation 3-9 states the meaning of expansion (similar to the usual subset operator) and formalizes the definition of the *Ideas* nest from Section 3.3.3.

$$\forall x, y : \left(\begin{array}{l} \text{Expand}(x, y) \\ \Leftrightarrow \forall a : (a \in x \Rightarrow a \in y) \end{array} \right) \quad \forall x : \left(\begin{array}{l} x \in \text{Nests} \Rightarrow \\ \text{Expand}(x, \text{Ideas}) \end{array} \right)$$

Equation 3-9. Rules related to nest expansion

3.4.3. Metatype Compatibility Problem

Our metamodel so far still lacks the constraints necessary to prevent the situation depicted in Figure 3-32, where a ternary relation extends a binary one.

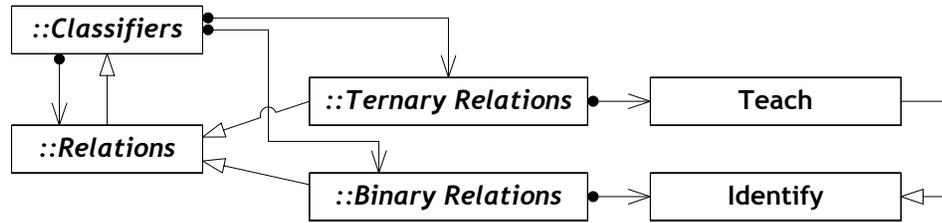


Figure 3-32. Example of incompatible metatypes

While in this example ternary and binary relations would have conflicting constraints on the nature of their members that would force the *Teach* relation to be empty, this is not always the case. To state the problem more generally, let us define metatypes as classifiers whose instances are classifiers, or equivalently as a classifier that expands *Classifiers* (assuming that *Expand* is reflexive) (Equation 3-10).

$$\begin{aligned} (m \in \text{Metatypes}) &\Leftrightarrow (m \in \text{Classifiers} \wedge \forall c : (c \in m \Rightarrow c \in \text{Classifiers})) \\ &\Leftrightarrow (m \in \text{Classifiers} \wedge \text{Expand}(m, \text{Classifiers})) \end{aligned}$$

Equation 3-10. Definition of metatypes

Then sometimes the instances of two metatypes are simply incompatible and should not be allowed to extend each other – Chapter 4 provides some more examples. Is there some general constraint we can enforce to invalidate these undesirable extensions?

A way to sidestep the problem completely would be to have incompatible metatypes not extending *Classifiers*, but rather introducing a new class-like concept with a matching new extension-like relation. In the example above, this

would amount to removing all the extension relationships and defining new relations called *Extend-binary-relation* and *Extend-ternary-relation* that would only apply to instances of the appropriate relation metatypes. This approach leads to a proliferation of almost-but-not-quite-the-same relation definitions and fails to express the commonalities in the meaning of the various extension relations. It is not appropriate for an ontology whose purpose is to integrate disparate meta-models into a common framework.

The next idea that comes to mind is simply to forbid extension relationships between classifiers with different metatypes. This would mean that if C_1 extends C_2 , then C_1 and C_2 must have the same metatypes. However, such a rule would prohibit the situation [Hay02b] shown in Figure 3-33, which clearly makes sense.

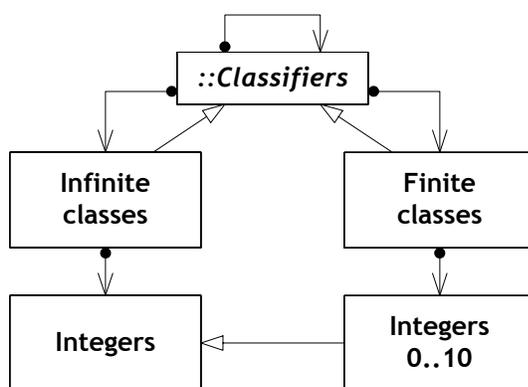


Figure 3-33. Example of valid cross-metatype inheritance

Requiring the metatypes to match exactly is too strong. Taking a look at UML, we see that the equivalent rule in [OMG01], p. 2-61, Section 2.5.3.20, paragraph 5 is:

A GeneralizableElement may only be a child of a GeneralizableElement of the same kind.

This works fine for the UML M_1 model (with metatypes from M_2), but would fail if the example above were restated in UML, since *Finite classes* does not extend *Infinite classes*. There are even valid examples where the metatypes' extension

relationship goes in the inverse direction of their member classifiers' (e.g., Figure 3-34).

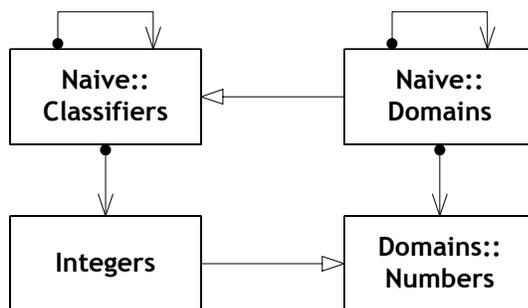


Figure 3-34. Metatypes inverting instances' extension

3.4.4. Metatype Constraint

The solution to the metatype compatibility problem adopted in the Braque metamodel is to differentiate between classifier extension and mere set expansion. Expansion is a purely structural feature that only implies that the members of the child will also be in the parent. It can be expressed between any two nests. Extension is the stronger statement that the child classifier is a specialization of the parent classifier. Not only can the child's members be used wherever instances of the parent are expected (the Liskov Substitution Principle [Lis88]), but the intent of the child classifier is a refinement of the parent's intent.

For most classifiers, the distinction is slim with no concrete consequences in the model. For metatypes, though, we can take advantage of this difference by requiring that, in an extension relationship, the classifiers' metatypes roughly match, up to extension. Equation 3-12 expresses the constraint, assuming that the extension relation is transitive and reflexive (Equation 3-11).

$$\begin{aligned} \forall x, y, z : (\text{Extend}(x, y) \wedge \text{Extend}(y, z)) &\Rightarrow (\text{Extend}(x, z)) \\ \forall c : (c \in \text{Classifiers}) &\Rightarrow (\text{Extend}(c, c)) \end{aligned}$$

Equation 3-11. Extension is transitive and reflexive

$$\begin{aligned} & \forall c_1, c_2 : (\text{Extend}(c_1, c_2)) \\ & \Rightarrow \left(\begin{array}{l} \forall m_1 : \left(\begin{array}{l} c_1 \in m_1 \wedge \\ m_1 \in \text{Metatypes} \end{array} \right) \\ \Rightarrow \exists m_2 : \left(\begin{array}{l} c_2 \in m_2 \wedge \\ \text{Extend}(m_1, m_2) \end{array} \right) \\ \wedge \\ \forall m_2 : \left(\begin{array}{l} c_2 \in m_2 \wedge \\ m_2 \in \text{Metatypes} \end{array} \right) \\ \Rightarrow \exists m_1 : \left(\begin{array}{l} c_1 \in m_1 \wedge \\ \text{Extend}(m_2, m_1) \end{array} \right) \end{array} \right) \end{aligned}$$

Equation 3-12. The Braque metatype constraint

The effect of the metatype constraint is to partition metatypes into connected components based on the *Extend* relation. Extension is allowed only when both classes belong to the same connected components. *Expand* relationships can be used as a kind of firebreak in the metatype hierarchy, separating it into independent regions while preserving the transitive promotion of instances into parent types.

The Braque metatype constraint is inspired by the UML generalization rule quoted above, but there are some important differences. First, since UML enforces single classification, its rule can refer to a class' unique metatype, whereas in Braque a class may have any number of metatypes. For this reason, the metatype constraint must be satisfied for every metatype of each classifier participating in the extension relationship. Second, for each metatype of one classifier, it is sufficient to find any one metatype of the other classifier that is extended by the first one. This relaxes the UML rule in two ways. The child classifier's metatype does not need to be a subtype of the parent's metatype – instead, they must share a common supertype. We also get to pick any of the child classifier's metatypes to satisfy the condition, which could be any parent (by extension or expansion) of the “primary” one. These relaxations dispose of the objections levelled against the UML rule.

3.4.5. Constraining Naïve Metatypes

Before changing our ontology to take advantage of this new rule, let us introduce a new shorthand graphical notation to express the *Expand* relationship (Figure 3-35).



Figure 3-35. Expand relationship shorthand

We can now model the relation topics in our ontology as shown in Figure 3-36. Note that with *Binary Relations* expanding *Relations*, rather than extending it, the situation of Figure 3-32 is prevented by the metatype constraint, since there is no common metatype extended by *Binary Relations* and expanded by *Ternary Relations*.

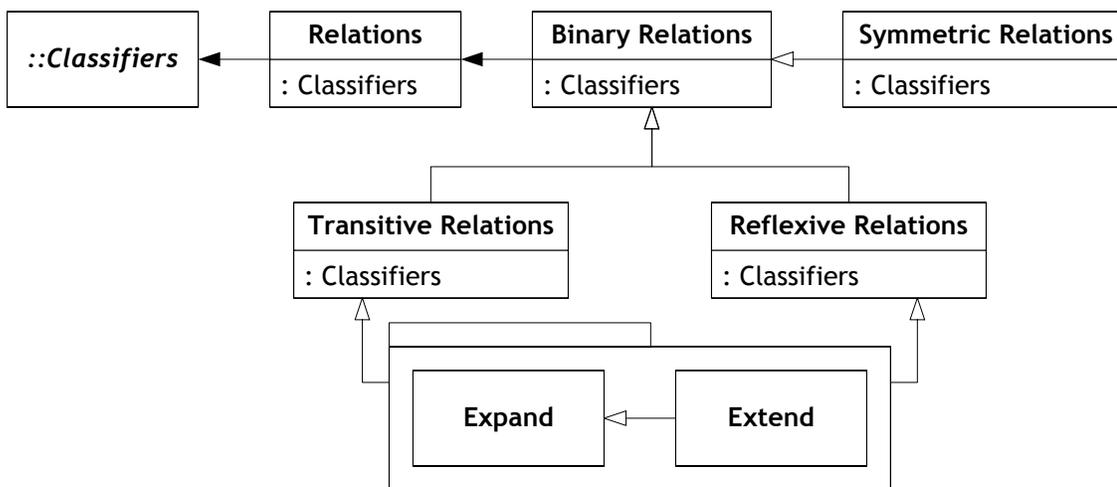


Figure 3-36. Improved model of naïve relations

Another interesting question is whether the *Transitive Relations* metatype should extend or expand *Binary Relations*. If we put in a firebreak with *Expand*, then transitive relations will be unable to extend intransitive ones, and vice-versa. The first should clearly be possible, but can we find a valid example of an intransitive relation that extends a transitive one? Yes: define R and R_Δ as per

Equation 3-13. R is transitive while R_Δ is not, yet R_Δ could be said to extend R since all member pairs of R_Δ are also members of R .

$$\begin{aligned} R(x, y) &\equiv x < y \\ R_\Delta(x, y) &\equiv y - \Delta < x < y, \Delta > 0 \end{aligned}$$

Equation 3-13. Intransitive relation extending a transitive one

Transitive relations are thus truly a refinement of binary relations, since their instances can cross-specialize. Similar arguments hold for symmetric and reflexive relations. Equation 3-14 expresses my hypothesis that if two metatypes are extension-compatible in one direction then the reverse direction must be valid as well. This hypothesis is plausible but unproven. Should it turn out to be false, the metatype constraint would have to be revised since, as written, it is independent of the direction of extension between metatypes.

$$\begin{aligned} \exists m_1, m_2, c_1, c_2 : &\left(\begin{array}{l} c_1 \in m_1 \wedge c_2 \in m_2 \\ \wedge \text{Extend}(c_1, c_2) \end{array} \right) \\ \Rightarrow \exists d_1, d_2 : &\left(\begin{array}{l} d_1 \in m_1 \wedge d_2 \in m_2 \\ \wedge \text{Extend}(d_2, d_1) \end{array} \right) \end{aligned}$$

Equation 3-14. Extension compatibility symmetry hypothesis

3.5. Issues of Logic

Since Braque is an information model, not just a data model, it's important to define the meaning of its structures. As noted in sections 3.2.1 and 3.2.2, atoms reify arbitrary things while nests reify relationships between things. When putting ideas into a nest, it's always a relationship between the reified things that is being asserted, not between the idea objects themselves. To assert a relationship between the reifying constructs (for example, their relationship to the referent, or to the underlying programming language's type system), we'd need to reify them in turn.

It is very important to understand that reifying a thing does not assert its existence. Reifying a unicorn doesn't mean that we believe in mythological creatures, and reifying the relationship "Peter has a M.Sc." does not make it true. This way,

we can talk about things we don't necessarily believe, or even assert that a relationship is not true. If every relationship in the model were always asserted, this last would result in a contradiction.

How does one make assertions, then? Braque does not enforce any specific mechanism or logic, but the structures supplied should be sufficient to build ontologies of truth. One simple approach would be to gather all trusted relationships into a single set, then limit further queries to only use information from that set. More complex ontologies might assign truth probabilities to relationships, or perform time-sensitive reasoning by considering validity intervals for assertions. All kinds of mechanisms can be implemented easily since universal referenceability makes it trivial to talk about relationships of relationships.

It is also worthwhile to discuss how membership reification fits into logical interpretation. Asserting a relationship does not automatically assert its reified membership relationships and vice-versa; the idea of a statement being true is separate from assertions about its composition. While Braque ensures that the membership reifications share the lifecycle of their parent nest, each relationship can be asserted independently of the others.

Now that we understand the structure and meaning of the Braque metamodel we can describe how to use it to integrate other metamodels.

4. Integration

This chapter proposes mappings that integrate three currently popular semantic web metamodels into the Braque metamodel, and in so doing evaluates the expressive power of Braque. The mappings preserve all the original information and can easily be reversed to recover the original model.²⁴ However, exporting information to a metamodel other than the one it was originally imported from goes beyond integration and is therefore beyond the scope of this thesis. Such a feature would enable translation between metamodels using Braque as a middle ground and is an obvious target for future development.

The mappings exclude all schema and constraint information, since the NUO does not yet include such facilities. They also exclude (natural) language scoping constructs—though they appear in all three metamodels being integrated—since they are difficult to align and not critical to the information integration effort.

Sections 4.1, 4.2 and 4.3 introduce the three mappings, while section 4.4 provides a larger example of integration. For easy reference, Appendix B collects the ontological structures needed by the mappings that are introduced in this chapter.

4.1. Extensible Markup Language

The Extensible Markup Language (XML) [BPS00] is a set of rules for defining textual formats for structured data storage. It comes from a tradition of markup languages (such as HTML [RHJ99]), themselves designed according to the rules of XML's precursor SGML [ISO86]. XML's rules are a vast simplification of SGML, which is widely believed to be responsible for XML's widespread popularity.

While XML imposes some minimum well-formedness rules on its markup languages, it does not specify any common semantics for them [Cov98]. This means

²⁴ Minor syntactical details of the serialization formats (such as white space) are purposely discarded, since the goal is a mapping between models, not their representations.

that not only does each dialect have its own vocabulary, but it also assigns its own interpretation to XML's primitive features. In terms of the IMI Reference Model, XML does not (and cannot) define a generic object layer, so each application must make up its own.

This lack of semantics seriously hampers the integration of information from XML documents into a common object-oriented metamodel. In this thesis, I take the less ambitious route of only integrating the basic structure of XML documents without trying to decode its meaning. An application with dialect-specific knowledge can then transform this structural model into the information actually contained in the document. Other approaches to exposing the semantics of XML documents are summarized in Section 5.2.

4.1.1. Basic Structure of XML Documents

All XML documents follow a basic tree structure of nested elements intermixed with free-form text. The elements and text are totally ordered, and each element is unique—there's no way to escape the tree structure. Every element also has any number of attributes, each associating a simple string value to the element. Both elements and attributes are labeled. (A full treatment of the complex naming system is deferred to the next section.) Each element can have at most one attribute for any given label.

Based on this description, XML documents are normally represented as a labeled tree. Each vertex represents an element, and is linked by (directed) edges to its attribute values and children elements. The attribute edges get the attribute labels, since the values are always strings and don't need further labeling. What about children elements, though? Should the label go on the edge, indicating the role played by the child, or should it go on the element itself, indicating its type? Without further information (in the form of extra annotations or a schema), the decision is arbitrary. The mapping to Braque considers element labels as names

of element types, but could just as easily interpret them to be role types, used for classifying the membership relationships between an element and its children.

In any case, XML's tree-shaped structure fits the Braque metamodel perfectly. XML elements are nests with list characteristics (totally ordered and accepting duplicates²⁵). Elements contain other elements and strings. Attributes are binary relationships between the element and the assigned string value. Figure 4-1 demonstrates the result of mapping a small XML document into Braque, with a not uncommon liberal treatment of white space.

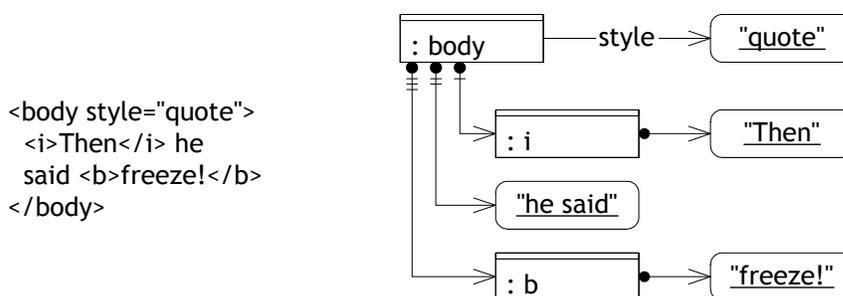


Figure 4-1. Mapping of sample XML document

The corresponding element and attribute types are defined in Figure 4-2. The types are instances of *Element Types* and *Attribute Types*, two metatypes that expand *Classifiers* and *Binary Relations*, respectively. While XML itself does not define type inheritance, it seems prudent to partition XML's element and attribute types from other classifiers and binary relations in preparation for the introduction of XML schema facilities in the future. All elements and attributes are also aggregated into the *Elements* and *Attributes* nests for convenience, as stated by Equation 4-1.

²⁵ Elements are unique, but the same string fragment may occur more than once inside an element.

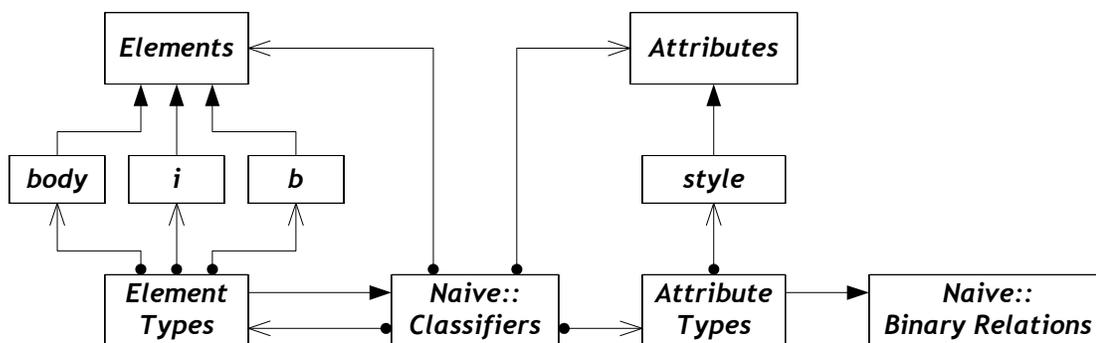


Figure 4-2. Sample XML element and attribute types

$$\begin{array}{ll} \forall t : (t \in \text{Element Types}) & \forall t : (t \in \text{Attribute Types}) \\ \Rightarrow (\text{Expand}(t, \text{Elements})) & \Rightarrow (\text{Expand}(t, \text{Attributes})) \end{array}$$

Equation 4-1. Aggregation of XML elements and attributes

Thanks to totally ordered nests, the basic structure of XML comes through very cleanly into Braque, and the mapping would be trivial were we to stop here. However, the surprisingly complex structure of XML names must also be mapped.

4.1.2. XML Names

Superficially, element and attribute names in XML appear to be quite simple: a simple string of characters denotes each type. Problems only start to appear when different people want to use the same tag name to mean different things, since this can confuse an application that expects “<p>” to indicate an HTML paragraph instead of a personal contact entry in an address book. The usual solution—to partition names into disjoint namespaces—is the one developed in the XML Namespaces recommendation [BHL99]. While this recommendation is separate from the base XML standard, it is very commonly used and any mapping of XML that does not support XML Namespaces would not be worth using. This section builds one possible model of XML Namespaces and hooks it up to the element and attribute types created above.

XML names are not simple strings; they have internal structure. First, the same string can be used to denote both an element type and an attribute type without

conflict. The name represented by that string depends on whether it was encountered in an element or an attribute context. Second, a name can be simple or qualified. A simple name is represented by a simple string and the aforementioned context. A qualified name is a pair consisting of a simple string—the “local part”—and a URI that identifies the namespace. The model so far is shown in Figure 4-3.

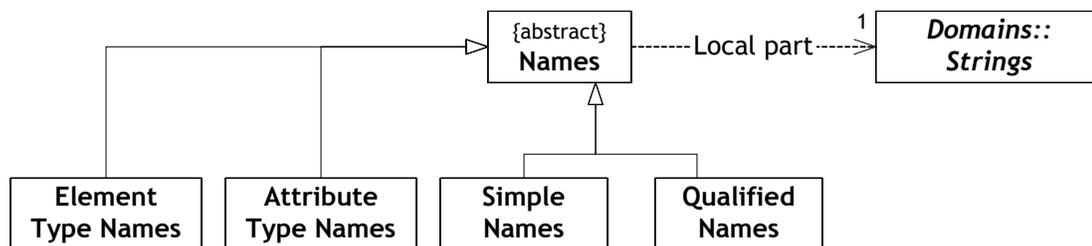


Figure 4-3. Four kinds of XML names

4.1.3. XML Namespaces

We must now model namespaces. Instead of just associating qualified names to their namespace URI, this ontology models a namespace as a concept in its own right, a mildly controversial approach since not everybody agrees that XML namespaces “exist” [Bou00]. A top-level namespace, then, is a set of qualified names identified by a URI (Figure 4-4).



Figure 4-4. Top level XML namespaces

What about simple names? Simple element type names are not considered to be in any namespace. Simple attribute type names, on the other hand, belong to a namespace identified by the type of the element they are decorating. For example, in “<p a='1'>” and “<b a='x'>”, there are two separate attribute types called “a”. Thus, each simple attribute type name belongs to a local attribute namespace (Figure 4-5).



Figure 4-5. Local attribute XML namespaces

Through generalization, we arrive at the generic concept of an XML namespace (Figure 4-6) and a statement about the distinguishing characteristics of XML names (Equation 4-2).

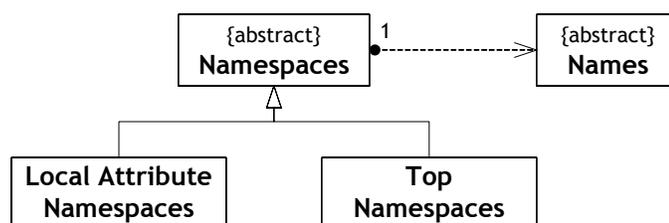


Figure 4-6. XML namespaces

$$\forall x, y : \left(\left(\left(\left(\left(\begin{array}{l} t \in \{ \text{Element Type Names,} \\ \text{Attribute Type Names} \} \\ n \in \text{Namespaces} \wedge \\ x \in t \wedge y \in t \wedge \\ x \in n \wedge y \in n \wedge \\ \text{Local part}(x, s) \wedge \\ \text{Local part}(y, s) \end{array} \right) \right) \right) \right) \Rightarrow (x = y) \right)$$

Equation 4-2. XML names uniqueness constraint

Completing the ontology, qualified element type names identify element types, while simple element type names are potentially ambiguous and merely denote them. Both simple and qualified attribute type names identify attribute types. The full XML integration model is reproduced in Appendix B, and Section 4.4 explores the mapping of a larger sample document.

4.2. Resource Description Framework

The Resource Description Framework (RDF) is a simple metamodel for defining and exchanging information on the semantic web. It is championed by Tim

Berners-Lee and is hence likely to remain relevant to knowledge integration efforts despite its many problems.

RDF is still under development. The base model and syntax recommendation [LS99] has been available for a few years, but it is undergoing heavy revision [Bec02][KC02] by the RDF Core workgroup (<http://www.w3.org/2001/sw/RDFCore/>) and there are many outstanding issues still left to be resolved. The RDF Schema (RDFS) draft [BG02] specifies a small upper ontology on top of RDF, but it too is a work in progress and has never been officially published. Finally, an RDF Model Theory [Hay02a] that formally defines the semantics of RDF and RDFS constructs is also under development.

Consequently, while the foundations of RDF are fairly solid and well understood, the more advanced features (that nonetheless belong to the object layer of the IMI Reference Model) are still very much a moving target. This section provides a full mapping from RDF and RDFS (minus constraints and language tagging) into the Braque metamodel according to all documents available at time of writing and supplemented by my own interpretation where necessary.

4.2.1. Basic Structure of RDF

At its core, RDF has a simple, domain-neutral metamodel. A model consists of an unordered set of statements. Each statement is a triple that relates a subject and an object through a predicate. A small sample model in NTriples format ([GB02] Section 3, work in progress as usual) is given in Figure 4-7. Other formats exist (including an XML serialization and a directed graph representation [KC02]) but are not of interest here.

```

<http://www.ideanest.com/research/Thesis.doc>
<http://purl.org/dc/elements/1.1/creator>
"Piotr Kaminski" .

<http://www.ideanest.com/research/Thesis.doc>
<http://purl.org/dc/elements/1.1/publisher>
_:uvic .

_:uvic
<http://purl.org/dc/elements/1.1/title>
"University of Victoria" .

```

Figure 4-7. Sample RDF model in NTriples format

The subject of each statement is a resource. The RDF conception of a resource is essentially that explained in Section 2.1.3, so a resource can be just about anything. In the first two statements above, the subject is a copy of this thesis document, while in the last statement the subject is the University of Victoria institution. Every resource in an RDF model either has a single global URI identifier (e.g., the first two subjects), or is a “blank” resource with unique identity but no identifier at all (e.g., the last subject). In a model, blank resources are identified only by the relationships they enter into. In the NTriples format, each blank resource gets a local identifier similar to a Braque label.

Each statement’s predicate describes the relationship between the subject and the object. It is also a resource and allowed to be blank. In the example above, the predicates were chosen from the standard Dublin Core Metadata Element Set (DCME) [WK+99]. DCME is not specific to RDF, so it recommends that the object of a “creator” or “publisher” predicate be the name of the corresponding entity. This recommendation was followed in the first statement, but in RDF it looks a little strange to have a string as the creator of a document. For this reason, the second and third statements employ an idiom more suited to RDF: the publisher is some blank resource whose title is a string.

Finally, the object of each statement is either a resource or a literal. RDF’s literals are structured objects, not just strings. A literal may have a language tag, and

can be interpreted either as a simple string or as an XML fragment.²⁶ A literal cannot be the subject of a statement and, at the moment, cannot be interpreted as a typed value (e.g., an integer) though that may soon change [HMS02].

The basic RDF metamodel is amenable to a straightforward mapping into Braque. Each statement is a binary relationship between the subject and the object; the predicate is the relationship type (i.e., the relation). A resource is normally mapped to an atom, unless it is used as a predicate, in which case it must be mapped to a relation nest. Nothing more needs to be done for blank resources, though for convenience the equivalent idea may be labelled with the same name as was used in the NTriples document. Resources identified by a URI are expressed as described in Section 3.3.7. Literals are either strings, or are mapped into XML structures as per Section 4.1. Literal language scoping is ignored.²⁷

The mapping of the sample model according to these rules is shown in Figure 4-8.

²⁶ The best reference on this interpretation of literals is [Bec02], since the structure was not yet formalized at the time of this writing.

²⁷ It is very strange that RDF makes language a property of the literal itself. After all, the string itself does not have a language—only through use does it gain meaning and bring the language into question. It seems to me that it is RDF triples that should be scoped by language, not literals.

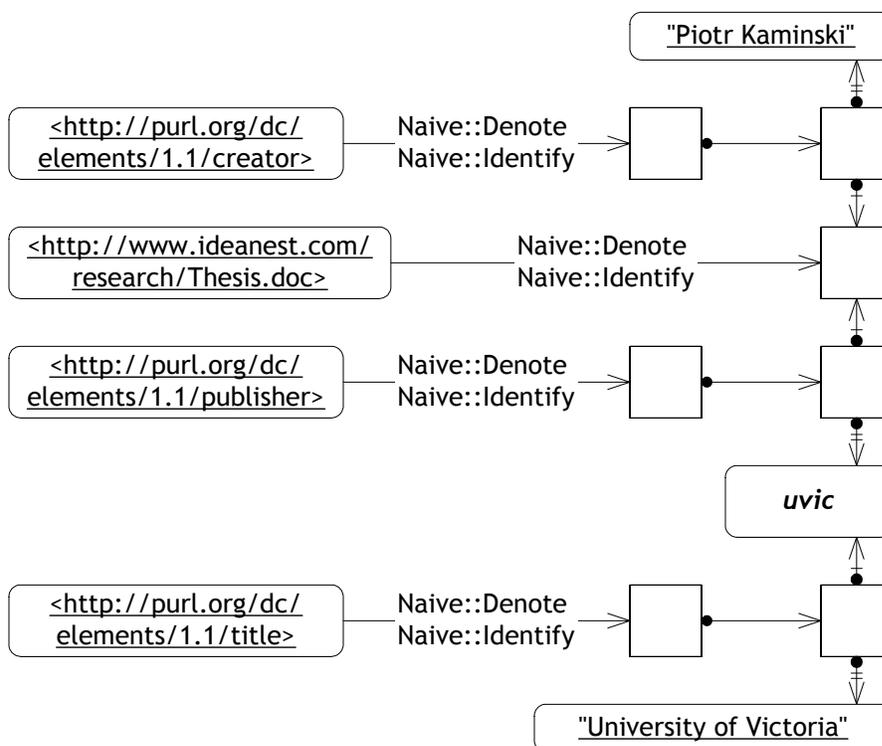


Figure 4-8. Mapping of sample RDF model

4.2.2. RDF Types, Properties and Values

Apart from the fundamental modeling mechanism, RDF also introduces a few basic concepts as part of a minimal upper ontology. In theory, the basic structural mapping is sufficient to translate this ontology into Braque. In practice, though, it is useful to embed RDF's ontology into Braque's NUO to facilitate integration between metamodels. This section only treats basic typing and values; more advanced features and RDFS extensions to this ontology are covered in later sections.

RDF introduces only two concepts to provide basic typing: *Property* and *type*. *type* is a resource used as the predicate when stating the type (class) of another resource. *Property* is the type of all resources that can be used as predicates. As expected, *type* is of type *Property*. *Property* does not have a type at this stage, since the *Class* concept is only introduced in RDFS.

RDF also introduces a vaguely defined *value* property that is meant to “identify the principal value (usually a string) of a property when the property value is a structured resource.” [BG02]

Embedding these concepts in the NUO is fairly straightforward. To match the structural mapping, *Property* is an extension of *Binary Relations*. It is an extension rather than an expansion since there is no reason to segregate RDF properties from other binary relations and prohibit their extension outside of the RDF ontology.

The *type* relation is a little trickier, since in Braque *type* is indicated through nest membership. However, we can take advantage of membership reification to align the two concepts. The *Member* relation has a container-member pair for each member of each nest in the model, including instances of classifiers. Inverting this relation, we obtain pairs of member-container. The *type* relation is then an extension of this inverted relation, since it links instances to their classes.

The *value* relation seems to be some subset of the inverse of the *Represent* relation. (This is arguable, but the definition of *Represent* is fairly broad.)

The basic embedding is illustrated in Figure 4-9. Both the RDF *type* and the implied Braque membership relationships are illustrated, though we will forgo the latter from now on. The diagram also leaves out the full URIs assigned to each RDF resource to increase readability; the complete mapping identifies each of the RDF(S) elements with its published URI.

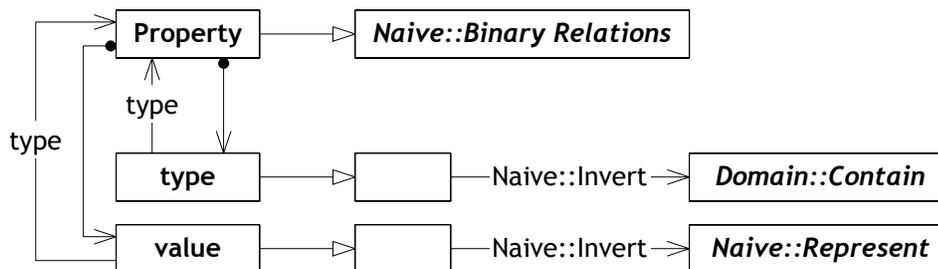


Figure 4-9. Basic RDF ontology embedding

Note that this is strictly an embedding of RDF into the NUO; it does not map naïve concepts back to their RDF equivalents. This reverse mapping is outside the scope of this thesis, but we can nonetheless demonstrate that it would be easy to add in at this level. First, we would need to merge *Property* and *Binary Relations*, which could be done by having each extend the other. Second, we would need to strengthen the equivalency between *type* and classifier membership, perhaps with a constraint like that presented in Equation 4-3.

$$\forall x, y : (\text{type}(x, y)) \\ \Leftrightarrow \left(\begin{array}{l} y \in \text{Classifiers} \\ \wedge x \in y \end{array} \right)$$

Equation 4-3. Strong RDF type mapping constraint

Creating reverse mappings increases in difficulty with the complexity of the foreign ontology, so we will not provide any further examples.

4.2.3. RDFS Classes, Hierarchies and Indicators

RDFS builds on RDF's simplistic notion of typing by introducing the *Class* concept. The type of every resource must be an instance of *Class* (that is, a resource of type *Class*). For example, since *type* is of type *Property*, *Property* is of type *Class*. *Class* is itself of type *Class*, giving RDFS an unstratified type model. RDFS also insists that every resource must be the instance of at least one *Class*, so it introduces *Resource*, the class of all resources. Finally, the class *Literal* is the class of all literals. This last class is rather useless, since RDF cannot put a literal in the subject position of a statement to assign it the *Literal* type. (For a more detailed investigation of this problem, see [Hay02a] Section 3.3.1.)

The mapping is, once again, straightforward. *Resource* is an extension of *Ideas*, *Class* of *Classifiers* and *Literal* of *Strings*. Extension is preferred to expansion for the same reason as given in the previous section. Figure 4-10 shows this part of the RDFS ontology and its embedding in the NUO.

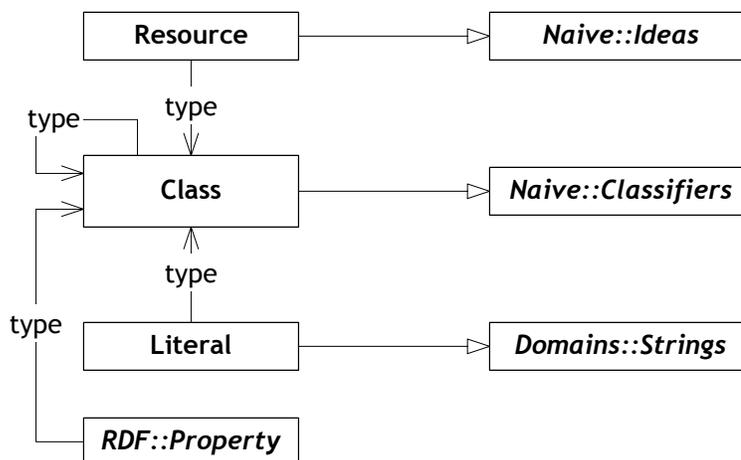


Figure 4-10. RDFS classes embedding

RDFS also allows one to establish refinement relationships between classes with the *subClassOf* property. Its semantics match those of *Extend* with the exception of the metatype constraint: RDFS allows any class to extend any other. We can still say that *subClassOf* extends *Extend*, since the lack of a concept of expansion in RDFS means that the metatype constraint becomes irrelevant in this context. We must also add the RDFS rule that *Resource* is the top-level superclass of all classes (Equation 4-4).

$$\forall x : \left(x \in \text{Class} \Rightarrow \text{subClassOf}(x, \text{Resource}) \right)$$

Equation 4-4. All RDFS classes extend Resource

subClassOf relationships can only be used between classes. To indicate refinement of relations, RDFS employs a separate *subPropertyOf* property, presumably to ensure that relations cannot extend classes and vice-versa. Since Braque handles this problem more elegantly by enforcing the metatype constraint, we can consider *subPropertyOf* as an extension of the *Extend* relation, which in Braque applies to relations as well as to classes.

RDFS also has some auxiliary properties: *comment* links a resource to a description, *label* links a resource to a human-readable name, and *seeAlso* links a resource to a related one. These can be neatly tied to the indication hierarchy of the NUO,

though the simplistic *comment* and *label* relationships that connect strings directly with the indicated object need to be expanded to include an anonymous mediating atom that reifies the description or name (Equation 4-5).

$$\begin{array}{ll} \forall x, y : (\text{comment}(x, y)) & \forall x, y : (\text{label}(x, y)) \\ \Rightarrow \exists z : \left(\begin{array}{l} \text{represent}(z, x) \wedge \\ \text{represent}(y, z) \end{array} \right) & \Rightarrow \exists z : \left(\begin{array}{l} \text{denote}(z, x) \wedge \\ \text{represent}(y, z) \end{array} \right) \end{array}$$

Equation 4-5. Translation of RDF comments and labels

Figure 4-11 shows the RDFS properties and their embedding into the NUO.

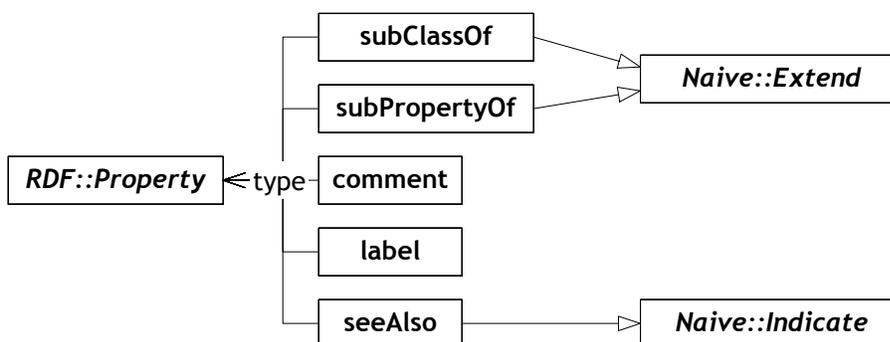


Figure 4-11. RDFS properties embedding

RDFS also defines *domain* and *range* properties, used to constrain the valid endpoints of binary relations, but as mentioned in the chapter's introduction their mapping is outside the scope of this thesis.

4.2.4. Containers

Models often need to describe collections of items. In RDF, the easiest way to do this is to have a group of statements sharing a subject and predicate while enumerating the contents of the collection in the statements' objects. Figure 4-12 shows a sample RDF model that employs this technique, and Figure 4-13 its mapping into a Braque model.

```

_:Piotr <http://xmlns.com/foaf/0.1/mbox> <mailto:piotr@ideanest.com> .
_:Piotr <http://xmlns.com/foaf/0.1/mbox> <mailto:pkaminsk@csc.uvic.ca> .
_:Piotr <http://xmlns.com/foaf/0.1/mbox> <mailto:pkaminsk@uvic.ca> .
  
```

Figure 4-12. Repeated statements create an implicit collection

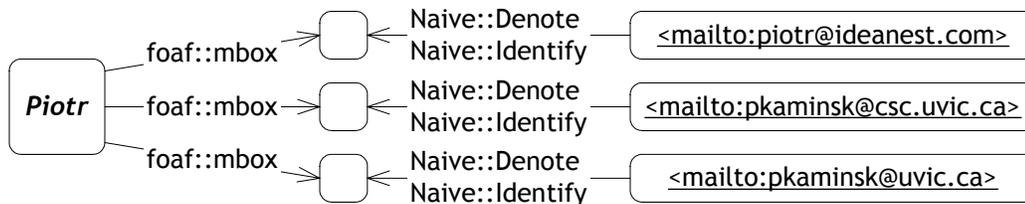


Figure 4-13. Mapping of repeated statements

While convenient and appropriate in many cases, this mechanism does not allow one to make statements about properties of the collection itself. In the above example, it is not clear whether all the email addresses are equivalent, which one is preferred (if any), etc. For this reason, RDF defines some standard container types that can be used to explicitly identify a collection as such, and RDFS completes the ontology. The separation seems rather arbitrary so this section will cover the combined RDF+RDFS container framework.

There are three container classes defined in RDF: *bags*, *sequences* and *alternative lists*. All three inherit from a common *Container* class; Figure 4-14 shows this hierarchy.

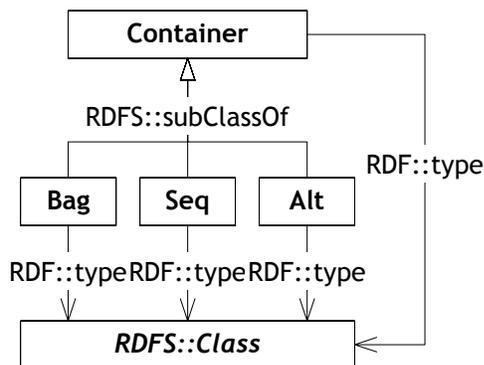


Figure 4-14. RDF container class hierarchy

The members of a *Bag* are unordered and can have duplicates. The members of a *Seq* are totally ordered but could still have duplicates. An *Alt* represents a list of equivalent alternatives, any one of which could be chosen without affecting a computation. There is a privileged “default” choice indicated with the predicate *_1* (see below), but the members are otherwise unordered, and duplicates are ap-

parently allowed. Since RDF employs multiple classification, a resource could be an instance of multiple container types simultaneously. It is not stated explicitly what would be the meaning of such a container, but the simplest assumption is that it will somehow satisfy the definitions of *all* its container types.

To fill its containers, RDF introduces an infinite number of ordinal membership properties named *_1*, *_2*, and so on, that are used as predicates to assign members to a container, ordered or not. They are all subproperties of a *member* property that can be used as a predicate if the members' order is irrelevant. All of these properties are instances of a special *ContainerMembershipProperty* subclass of *Property*, probably to allow them to be easily distinguished from other properties. Figure 4-15 shows the membership properties model.

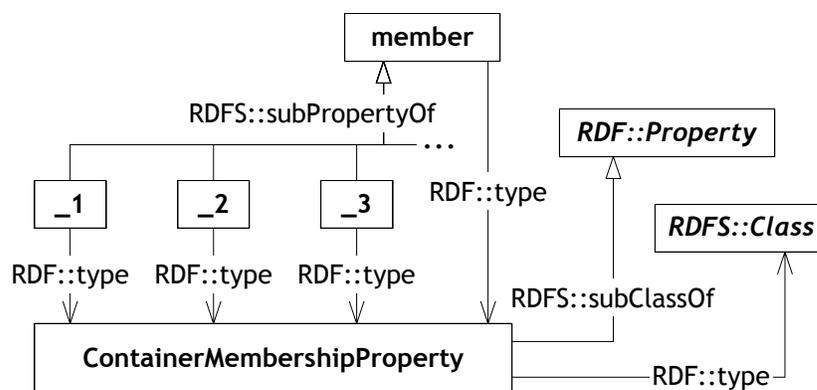


Figure 4-15. RDF membership properties model

We can now rewrite the example of Figure 4-12 to use a container to hold the various email addresses. Since all my addresses are funnelled to one mailbox they can be considered equivalent, so the *Alt* container is appropriate. The *piotr@ideanest.com* address is preferred²⁸ so it should be the default choice. The resulting triples are displayed in Figure 4-16.

²⁸ It is likely to remain valid longer than the others, since I own the domain name.

```

_:Piotr <http://xmlns.com/foaf/0.1/mbox> _:emails .

_:emails
<http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
<http://www.w3.org/1999/02/22-rdf-syntax-ns#Alt> .

_:emails <http://www.w3.org/1999/02/22-rdf-syntax-ns#_1> <mailto:piotr@ideanest.com> .
_:emails <http://www.w3.org/1999/02/22-rdf-syntax-ns#_2> <mailto:pkaminsk@csc.uvic.ca> .
_:emails <http://www.w3.org/1999/02/22-rdf-syntax-ns#_3> <mailto:pkaminsk@uvic.ca> .

```

Figure 4-16. Explicit container holds a collection

The definitions of RDF containers have some loopholes. For example, the RDF Model Theory cannot specify that, even if ordinal predicates are used in a *Bag*, the order does not matter, or that only the *_1* predicate matters in an *Alt*. Models are allowed to skip ordinals, and you are not allowed to assume that you ever know all the members of a container. It is also not (yet) defined what it means to have a *member* predicate in a *Seq* or to have the same ordinal predicate used twice for a container, but the RDFS constraint ontology does not have constructs powerful enough to forbid these possibilities. Overall, RDF provides an excellent demonstration of just how ugly it is to try to define collections within a labelled directed graph.

4.2.5. Containers Embedding

Braque nests are much richer than RDF containers, so the structural embedding does not present a problem.²⁹ *Container* is clearly an extension of *Nests*, while *member* is an extension of *Member* (Figure 4-17). Note that since RDF's ordinal properties are extensions of *member* their instances become elements of *Member* as well.

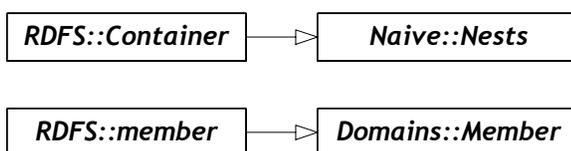


Figure 4-17. RDF containers embedding

²⁹ Reversing this mapping would be much more difficult.

This is sufficient to replicate the semantics of *Bag*. For the other two container types, we need to ensure that the RDF ordinal relations are ordered. Since conveniently they are already instances of the *ContainerMembershipProperty* class, we simply impose a partial order on this nest. The ordinal relations are ordered as per the indices in their names, while the *member* relation is unordered. We can now use the following constraints to impose the order of the relations onto the membership relationships for instances of *Seq* (Equation 4-6) and *Alt* (Equation 4-7), which induces an order on the members. Note that we cannot impose the order directly on the members since there might be duplicates, so we would not know which copy we were referring to. (The operator “ $<_N$ ” is the partial order on the members of nest N .)

$$\begin{aligned} & \forall c, r_1, r_2 : \left(c \in \text{Seq} \wedge r_1 \in \text{member} \wedge r_2 \in \text{member} \wedge \right. \\ & \quad \left. r_1 \neq r_2 \wedge r_1[1] = c \wedge r_2[1] = c \right) \\ \Rightarrow & \left(\exists! s_1, s_2 : \left(\begin{array}{l} s_1 \in \text{ContainerMembershipProperty} \setminus \{\text{member}\} \wedge \\ s_2 \in \text{ContainerMembershipProperty} \setminus \{\text{member}\} \wedge \\ r_1 \in s_1 \wedge r_2 \in s_2 \end{array} \right) \right) \\ \Rightarrow & \left((r_1 <_{\text{Member}} r_2) \Leftrightarrow (s_1 <_{\text{ContainerMembershipProperty}} s_2) \right) \end{aligned}$$

Equation 4-6. RDF Seq embedding constraint

$$\begin{aligned} & \forall c, r_1, r_2 : \left(c \in \text{Alt} \wedge r_1 \in _1 \wedge r_2 \in \text{member} \setminus _1 \wedge \right. \\ & \quad \left. r_1[1] = c \wedge r_2[1] = c \right) \\ \Rightarrow & (r_1 <_{\text{Member}} r_2) \end{aligned}$$

Equation 4-7. RDF Alt embedding constraint

Based on these constraints, Figure 4-18 shows the mapping into Braque of the *Alt* container described in Figure 4-16.

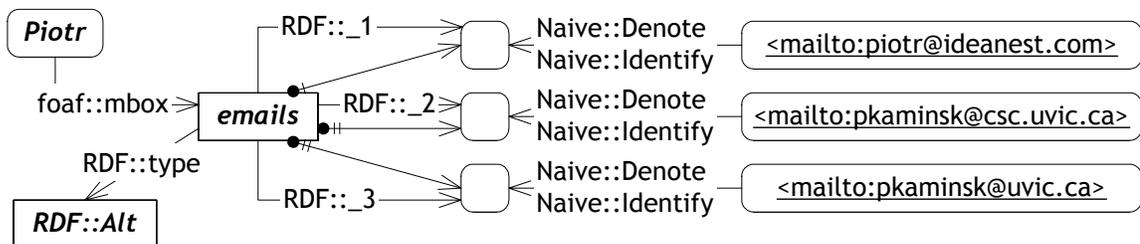


Figure 4-18. Braque model of explicit RDF container

While the embeddings make a perfect translation of the structure, the RDF container semantics are not precisely replicated. RDF is a metamodel that makes the open-world assumption, so triples missing from a model are assumed to be “unknown” rather than “not possible”. As mentioned in the previous section, this implies that RDF containers are never closed: it is always possible to add more membership triples. Since Braque does not provide a global interpretation, it is up to the agent to realize that the meaning of a missing element differs between RDF containers and other nests. Section 6.2.1 proposes an extension to the Braque metamodel that would allow nests to be explicitly open or closed.

4.2.6. Statements, Statings and Reification

An RDF model is composed of statements, but we must dig a little deeper to achieve an accurate semantic mapping into Braque. An RDF document is a bag of “statings”, each stating being the assertion of a statement. While statements can only be distinguished according to their subject, predicate and object, each stating is distinguishable from all others even if it asserts the same statement. This lets RDF distinguish between different expressions of the same statement, most often to assign provenance or other source-specific properties.

Since RDF does not enable direct references to either statements or stating, another mechanism is needed to reify them. RDF defines a standard ontology for describing statings by listing their subject, predicate and object. Confusingly, statings are assigned the *Statement* type, and statements themselves are never reified. It is also not possible to specify *which* specific stating a *Statement* instance reifies, only specify the corresponding triple and assure that the stating is unique. This is a lazy approach to reification, where the reification is constructed manually by the user [New02].

The mapping of this lazy reification into Braque is a little strange, since Braque eagerly reifies every relationship in the model. To correctly implement the semantics of RDF reification, each instance of *Statement* must actually *be* a relation-

ship for the given predicate. To maintain eager reification, each RDF relationship must also be an instance of *Statement* with the proper description. This provides a strong connection between a stating and its reification that is missing from RDF.

The mapping takes advantage of Braque’s membership reification. First, consider the mapping from the reified stating to the relationship. The instance of *Statement* will be the subject of three relationships, one each to specify the subject, predicate and object of the stated triple.³⁰ The subject and object pairs specify the two members of the stating’s relationship and can be added directly into *Member* (with the proper order). The predicate pair also specifies containment, but it is inverted, since it relates the stating to its predicate whereas it should be the predicate that contains the stating.

Second, consider the mapping from a relationship to its reified stating. To make sure that all stating relationships are instances of *Statement*, we specify that all properties expand *Statement* (Equation 4-8).

$$\begin{aligned} \forall t : (t \in \text{Property}) \\ \Rightarrow (\text{Expand}(t, \text{Statement})) \end{aligned}$$

Equation 4-8. Properties contain statements

When all the statings are gathered, we can translate the three containment relationships back to the reified stating description properties, reversing the mapping above. As an extra benefit, the subject and object properties also act like roles (as per Section 3.3.4), completing the description of the stating. Equation 4-9 formalizes the bidirectional mapping between RDF statings and their reifications.

³⁰ RDF does not specify the meaning of the reified stating if one of these components is missing or duplicated, but RDFS is not expressive enough to eliminate this possibility.

$$\begin{aligned} & \forall t : (t \in \text{Statement}) \\ \Rightarrow \forall r_1, r_2, r_3 : & \Leftrightarrow \left(\begin{array}{l} r_1 \in \text{Member} \wedge r_1[1] = t \wedge \\ r_2 \in \text{Member} \wedge r_2[1] = t \wedge \\ r_1 <_{\text{Contain}} r_2 \wedge \\ r_3 \in \text{Member}^{-1} \wedge r_3[1] = t \end{array} \right) \\ & \left(\begin{array}{l} r_1 \in \text{subject} \wedge \\ r_2 \in \text{object} \wedge \\ r_3 \in \text{predicate} \end{array} \right) \end{aligned}$$

Equation 4-9. RDF reification constraint

Another important difference between RDF and Braque is that in Braque all unreified statings are automatically asserted: there is no way to write down a statement without declaring it true. On the other hand, reified statings are not considered to be asserted—only their descriptions are held to be true. To faithfully model this in Braque, we introduce a new classifier called *Assertion* that extends RDF's *Statement*. All statings read in from an RDF document are explicitly added to the *Assertion* class, while statings demanded by a reified description are only members of *Statement*. This maintains RDF's distinction between asserted and reified statings, while making it trivial to change their status—a basic task that RDF cannot easily perform.

4.3. Topic Maps

Topic Maps have their roots in indexes, glossaries and thesauri; they are structures for organizing metadata about existing resources. They were originally conceived as an SGML [ISO86] architectural form based on HyTime, culminating in the ISO Topic Maps standard [ISO99], commonly called HyTM. In an effort to ride the success of XML, Topic Maps were recast into an XML vocabulary called XTM [PM01] by an independent organization, which was then folded back into the ISO standard. Their tumultuous evolution is now continuing within the ISO subcommittee SC34, which is attempting to construct a two-layer model of the paradigm, as well as additional constraint and query languages.

Perhaps due to their roots as a markup language, the abstract metamodel for topic maps is not very well defined. Each new “standard” made changes to fundamental concepts, and even now there is no agreement on the exact meaning of many traditional topic map constructs. In this section, I base my interpretation loosely on the XTM 1.0 specification [PM01], a popular if somewhat controversial processing model [NB01], and the current rough drafts of the Reference Model [NB02] and Standard Application Model [GM02] produced by SC34. The details of the mapping will definitely need to be revised as the formal topic map models are developed, but the basic ideas seem stable enough to make an early attempt worthwhile.

Due to its rich structure and peculiar terminology, the topic maps metamodel is often misunderstood by people with a background in simpler metamodels, but seems to fit well into the worldview of librarians and other information workers. This section provides a full mapping of both the metamodel and the Topic Maps vocabulary into Braque, with the exception of some association template proposals that have not been universally adopted.

I use the Linear Topic Map (LTM) notation [Gar02] to provide examples of topic maps. Though the notation is not normative and unable to represent every feature of the metamodel, it has the advantage of being human-readable, unlike the XML vocabulary XTM. There is unfortunately no agreed-upon graphical representation of topic maps.

4.3.1. Basic Structure of Topic Maps

Topic Maps place a rare, even unique, emphasis on the distinction between a reification and its referent. In the Topic Maps lingo, a topic reifies its subject. Each topic reifies a single subject, and, optimally, each subject is reified by only one topic. Most other metamodels assume that this mapping is one-to-one and hence consider the two concepts interchangeable, but Topic Maps place great stock in carefully controlling the relationship between a topic and its subject (see next sec-

tion for details). A topic is roughly equivalent to an idea in Braque, an object in UML and a node in RDF, while a subject is an idea's referent or an RDF resource.

A topic map is a set of topics and associations. An association is a special kind of topic that reifies a relationship (which is a kind of subject). It is similar to a Braque nest, a UML link or an RDF statement, and should not be confused with a UML association, which models a whole class of links. Each association relates together a number of topics, where each topic plays some role in the association. An association's topics are unordered, and the same topic can play any role any number of times; the semantics are those of a bag of topic-role pairs. Naturally, the roles are themselves topics that can participate in any capacity in other associations.

Figure 4-19 shows an example of a very small topic map in LTM format. Each of the first two lines introduces a new topic with the given ID. The ID is like the label in Braque graphs: it represents the identity of the topic but does not appear in the model itself. The third line defines an association between the two topics. It says that there is an association of type *authorship* between *Piotr* and *Thesis*, where *Piotr* plays the role of *creator* and *Thesis* plays the role of *work*. Notice that since the association's elements are unordered, it is necessary to explicitly specify the role of each member. (This statement implicitly introduces the topics of *authorship*, *creator* and *work*, which are not further defined. It also introduces an anonymous association, which, although it is a topic, cannot be referred to later since it lacks an ID. This is merely a side effect of the LTM notation.)

```
[Piotr]
[Thesis]
authorship ( Piotr : creator, Thesis : work )
```

Figure 4-19. Sample topic map in LTM format

We can now map the basic Topic Maps structures into Braque. Topics become ideas—atoms for the most part. Associations become bag-type nests that contain the member topics. The association type is a classifier that holds its instance as-

sociations in the usual manner; we will have more to say about topic classification in Section 4.3.3. The roles of an association's members are assigned using the mechanism introduced in Section 3.3.4, though we must work around a small snag.

Topic map authors often consider a class and a role to be the same subject, which causes confusion in Braque since they are both modelled as classifiers but with different extents. To avoid this unpleasant overlap, all topics used as roles in a topic map are assumed to actually represent classes, and are assigned a doppelganger nest to take over their role responsibilities. The default role and the (assumed) classifier it is derived from are linked using an *Enact default* relationship. This ensures that their extents will not be confused, yet leaves the two nests linked with a relationship that allows one to be derived from the other. It is not a perfect solution, since a topic map that does not overlap classifiers and roles will still get all its roles split, but it is consistent and workable. Other approaches to the problem are explored in Appendix C.4.

With this mapping, the example above is transformed into the Braque model shown in Figure 4-20.

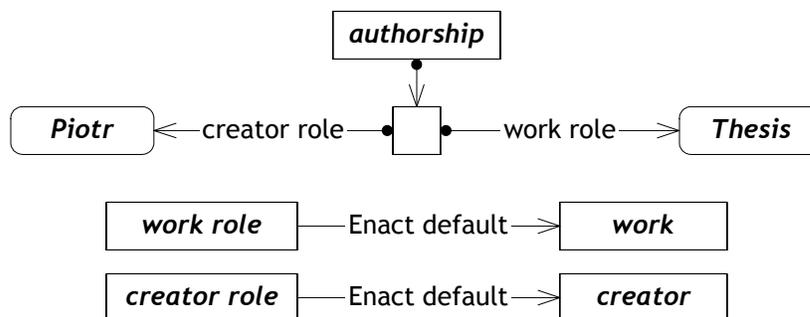


Figure 4-20. Mapping of topic map sample into Braque

4.3.2. Subject Identification

Associations relate topics, and we know that each topic reifies a single subject, but unless we can figure out what that subject is the relationship behind the as-

sociation will remain a mystery. The Topic Maps metamodel differentiates between two kinds of subjects, and offers three ways of pinpointing them.

Subjects are split into addressable and non-addressable ones. The exact definitions are a matter of some controversy, but essentially addressable subjects are supposed to be things that can be retrieved over the network, whereas non-addressable subjects cover the rest of the world. Addressable subjects are reified by specifying a *subject address* for the topic, usually a URI.³¹ Non-addressable subjects are reified with an additional level of indirection: the *subject identifier* is the address of a *subject indicator* resource that identifies the actual subject.³² For example, a subject identifier could be the URI of a web page (the subject indicator) that describes or illustrates the concept reified by a topic. A subject can have at most one address, and any number of identifiers.

While intuitively plausible, the dichotomy between addressable and non-addressable subjects, and the corresponding primitive difference between addresses and identifiers, is not present in other well-accepted standards. To wit:

- [BM+98a] clearly states that URIs can identify anything, including things that could not in any way be considered to live “in the network”.
- There is nothing in [BM+98a] prohibiting multiple URIs from identifying the same resource.
- [Fie99] insists that it’s never the resource itself that is retrieved over the network, but just a representation of it.
- [Mas02] proposes a “tdb:” (Thing Described By) URI scheme that can transform any subject identifier into a subject address, thus negating any difference between them.

³¹ The original topic map specification supported HyTime references, but most practical work today is done using URIs.

³² Note that subject identifiers and indicators do not directly correspond to the naïve upper ontology *Identify* and *Indicate* relations. The relationship is more complex, and explored later in this section.

The distinction between addressable and non-addressable subjects made by the Topic Maps metamodel is thus artificial and meaningless. A better way to divide the two concepts would be to say that addressable subjects have well-known addresses (URIs), while non-addressable subjects are best identified by description, though such a split is obviously rather subjective.

Be that as it may, subject addresses and identifiers are a part of the Topic Maps specification, and LTN allows both to be specified for a topic. As shown in Figure 4-21 below, a subject address is preceded with a “%” sign, and a subject identifier with an “@” sign. This example also demonstrates that the same resource (<http://www.ideanest.com/contact.html>) can be used as a subject and a subject indicator for two different topics (*ContactPage* and *Piotr*, respectively).

```
[Piotr      @"http://www.ideanest.com/contact.html"
          @"mailto:piotr@ideanest.com"]
[ContactPage %"http://www.ideanest.com/contact.html"]
authorship ( Piotr : creator, ContactPage : work )
```

Figure 4-21. Example of subject identifiers and indicators

Mapping subject addresses to Braque is simple: this is the same *Denote / Identify* relationship between a URI and an idea as in RDF. Subject identifiers, on the other hand, identify a subject indicator that in turn identifies the subject. Rather than introduce a new relation, we encode the extra level of indirection explicitly by modeling the intermediate resource. This resource may be anonymous, as for the email address in the example, or it may be the subject of another topic in the map, as for the contact page. Figure 4-22 shows the mapping of the example above into Braque according to these rules.

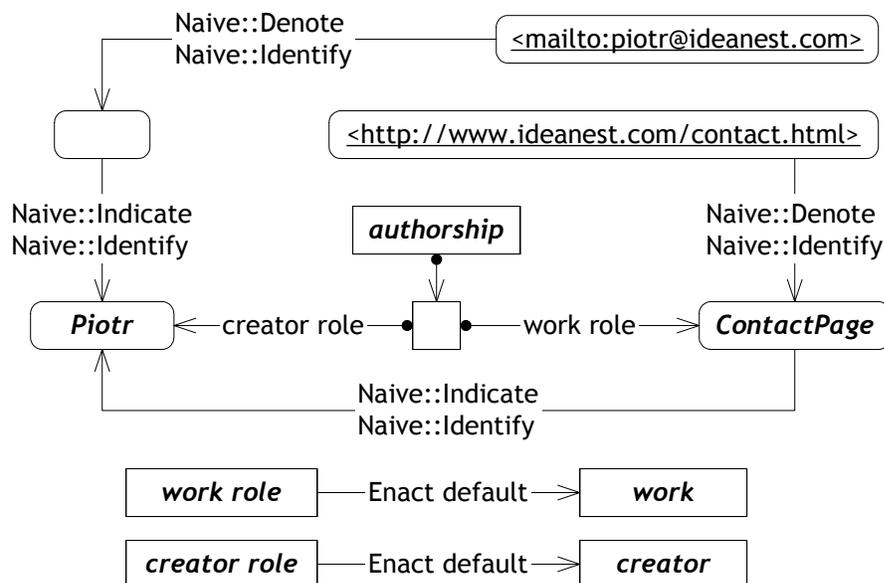


Figure 4-22. Mapping of identifiers and indicators into Braque

The third way to set a topic's subject is to embed it explicitly as a literal string. This technique is only allowed in certain situations, and hence will be explored later, in sections 4.3.5 and 4.3.6.

4.3.3. Class Ontology

The Topic Maps specification provides a small standard ontology for specifying subject classification and type generalization relationships, since neither is considered to be a metamodel primitive. Classification employs the *type-instance*³³ association type with the obvious role types, as used in Figure 4-23 to show that *Piotr* is an instance of *Person*.

```
[type-instance @"http://www.topicmaps.org/xtm/1.0/core.xtm#class-instance"]
[type          @"http://www.topicmaps.org/xtm/1.0/core.xtm#class"]
[instance      @"http://www.topicmaps.org/xtm/1.0/core.xtm#instance"]

[Person       %"http://xmlns.com/foaf/0.1/Person"]
[Piotr]
```

```
type-instance ( Person : type, Piotr : instance )
```

Figure 4-23. Specifying the type of a subject using an explicit association

³³ You may notice throughout this section that the identifiers use the word *class* instead of *type*. These identifiers were published in an older version of the specification, and carried forward unchanged to the current version to ease the transition.

This is rather long-winded, so LTM allows a shortcut notation to be used instead. Figure 4-24 is equivalent to Figure 4-23; the *type-instance* association is created implicitly by the LTM processor.

```
[Person %"http://xmlns.com/foaf/0.1/Person"]
[Piotr : Person]
```

Figure 4-24. Shortcut for specifying the type of a subject

A subject may be an instance of any number of types. There is no default “root type” (like RDFS’s *Resource*), so any subject with no explicitly assigned type simply does not have one. Furthermore, when using an explicit *type-instance* association to specify the type of an instance, if the roles used are other than precisely one each of *type* and *instance*, the standard interpretation is voided. At this time, Topic Maps lack a constraint language that could enforce this validity constraint.

While associations are technically “instances” of an association type, this classification relation is different from the one exposed above and cannot be expressed using an association. Were we to try, we would be led to an infinite regress of type-specification-associations, since for each *type-instance* association introduced we’d need to introduce another one to specify its type, and so on. To avoid this infinite regress (perhaps in an effort to keep the models finite?), Topic Maps uses a primitive single classification mechanism for associations that was already mapped into Braque in Section 4.3.1.

Specifying generalization relationships is done just like for classification, except that LTM does not provide a shortcut in this case. Figure 4-25 shows an example.

```
[supertype-subtype @"http://www.topicmaps.org/xtm/1.0/core.xtm#superclass-subclass"]
[supertype @"http://www.topicmaps.org/xtm/1.0/core.xtm#superclass"]
[subtype @"http://www.topicmaps.org/xtm/1.0/core.xtm#subclass"]

[Person %"http://xmlns.com/foaf/0.1/Person"]
[Student]

supertype-subtype ( Person : supertype, Student : subtype )
```

Figure 4-25. Specifying a generalization association

Mapping these association types into Braque is a simple matter of aligning the types and roles with those defined in the NUO. The *type-instance* association type is a subclass of the *Member* relation, while *supertype-subtype* is a subclass of *Extend*. These correspondences, and the mappings of the roles, are shown in Figure 4-26. We ignore the possibility of malformed associations, since their interpretation is unspecified anyway.

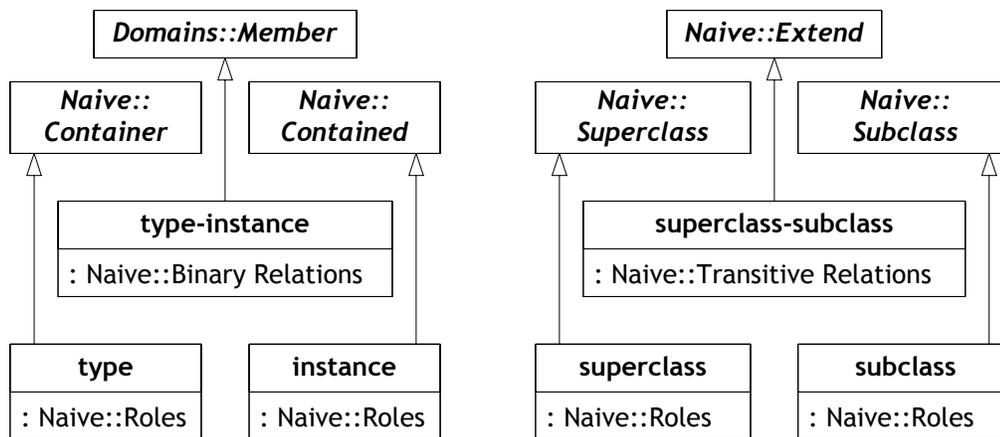


Figure 4-26. Topic Map classification and generalization embedding

4.3.4. Scopes

Topic maps can indicate the context in which a topic characteristic (an association for now, names and occurrences are added in the following sections) is valid by specifying a *scope*. But *caveat lector*: the specification of scopes is at an embryonic stage,³⁴ even considering the generally unfinished nature of Topic Maps. Thus, of necessity, the mapping proposed in this section is very tentative and explores a few possible interpretations.

Each association may have a scope that determines when the reified relationship holds true. A scope is defined by a set of subject *themes* that somehow circumscribe the applicability of the scoped relationship. Any subject can be used as a theme, but some common ones include: people, to scope assertions by opinion

³⁴ Scopes were excised from the RM [NB02], and in the SAM [GM02] an editor's note in the section on scopes states "This section needs to be reconsidered, then rewritten." See [Gra02] for a summary of the outstanding issues.

or point of view; languages, to scope names or other words by their natural language; and time and/or place, to scope by space-time location of the event. A simple agent can then set its “context” (a set of themes) so that it only sees the relevant associations. A more complex agent could compare or combine the information given in different scopes, depending on its needs.

In LTM, an association’s scope is set by listing its subject themes, separated from the association with a “/”. Figure 4-31 provides examples of scoping applied to the evaluation of various objects. In the first association, *Piotr* expresses the opinion that the *Thesis* is of *Excellent quality*. *Nigel* thought that the *Thesis* was only *Acceptable* before the defense (*BeforeDefense*), but changed his evaluation to *Excellent* after the defense (*AfterDefense*). He has also evaluated *Piotr* as an overall *Good* student. The last association asserts that *Piotr* is the *creator* of the *Thesis*, an unscoped (and hopefully uncontroversial) statement.

```
evaluation ( Thesis : object, Excellent : quality ) / Piotr
evaluation ( Thesis : object, Acceptable : quality ) / Nigel BeforeDefense
evaluation ( Thesis : object, Excellent : quality ) / Nigel AfterDefense
evaluation ( Piotr : object, Good : quality ) / Nigel
authorship ( Piotr : creator, Thesis : work )
```

Figure 4-27. Example of association scoping

When building the mapping into Braque, we must consider the fact that each scope is related to two different sets: the set of its themes, and the set of all topic characteristics (associations, for now) that are valid within it. We must decide which (if any) of these sets should become the scope’s extension, relegating the other to be implied by relationships attaching its elements to the scope. The Topic Maps documentation always talks about scopes as sets of themes, but since the question asked most often is probably “Is this characteristic in scope?” it seems more useful to consider a scope as a set of valid relationships. Each scope can then be interpreted as a kind of *Classifier* (actually, a free *Relation*) that imparts to its members the common property of validity in some given set of cir-

cumstances. Those circumstances are described by attaching themes to a scope using *Constrain* relationships.

Before we can map the topic map above into Braque we must also ponder what, if anything, to do with the last statement, which has no explicit scope. Topic Maps says that all such statements belong to an “unconstrained scope” that gathers all associations whose validity is unconstrained. We therefore introduce a distinguished *Unconstrained Scope* into the Topic Maps embedding, shown in Figure 4-28.

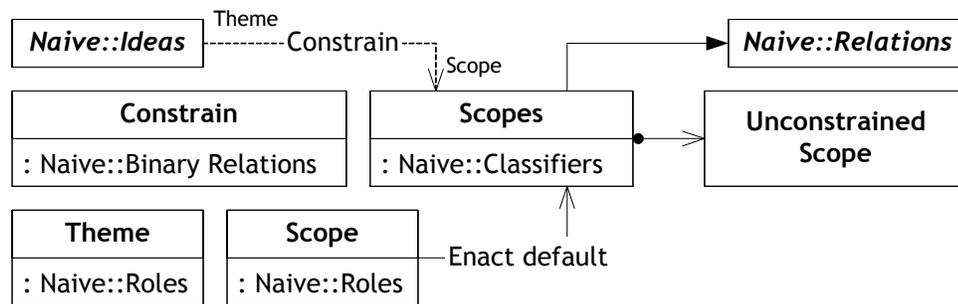


Figure 4-28. Embedding of Topic Maps scopes

Furthermore, since scopes that share the same set of themes are equivalent, we must introduce the identity constraint of Equation 4-10.

$$\forall s_1, s_2 : \left(s_1 \in \text{Scopes} \wedge s_2 \in \text{Scopes} \wedge \left(\forall t : (\text{Constrain}(t, s_1) \Leftrightarrow \text{Constrain}(t, s_2)) \right) \right) \Rightarrow (s_1 = s_2)$$

Equation 4-10. Scope identity constraint

With this machinery at our disposal, we can now map the topic map of Figure 4-27 into Braque, as shown in Figure 4-29.

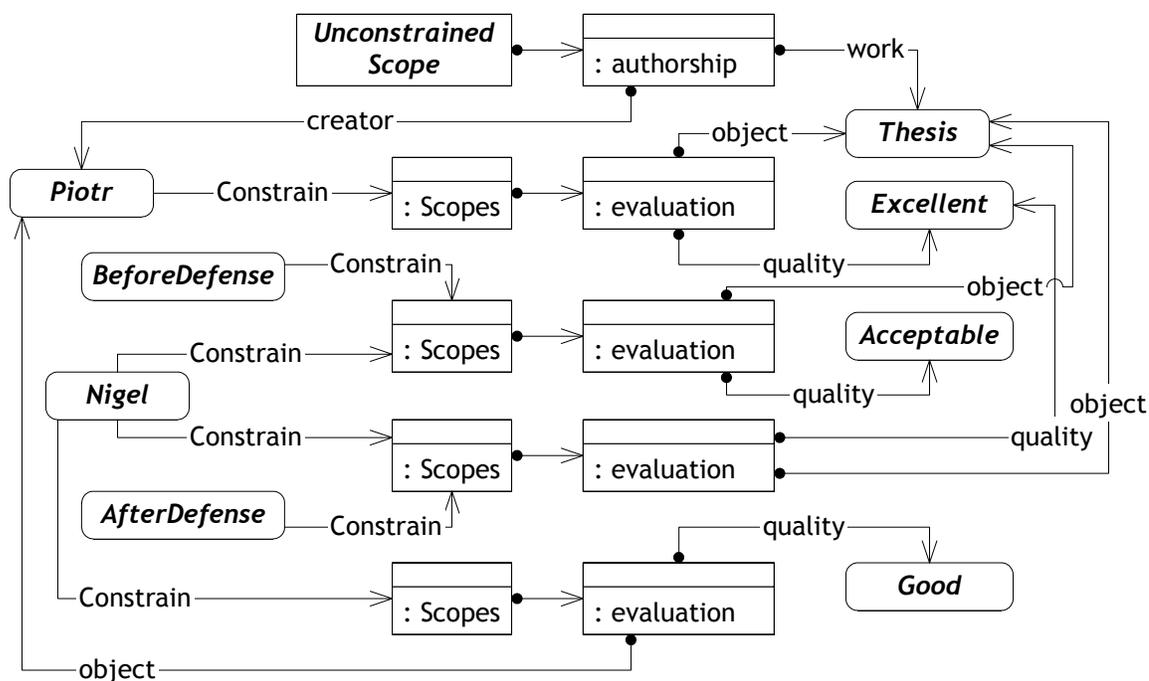


Figure 4-29. Example mapping of scopes into Braque

The interesting question now – and the one giving the writers of the Topic Maps standard the most trouble – is: what are the subsumption relationships between the scopes, based on their themes? If a topic characteristic is valid in some scope, does this imply that it is valid in other scopes as well?

Since validity is modeled by membership in a scope classifier, we can model entailment using *Extend* relationships. If every assertion in scope *A* must also be valid in scope *B*, then we can say that *A* extends *B*. Since every assertion in the unconstrained scope is always valid (that is, valid in any scope) by definition, we can say that the unconstrained scope extends every other scope (Equation 4-11). This makes *Unconstrained Scope* the bottom of the scope extension lattice.

$$\begin{aligned} \forall s : (s \in \text{Scopes}) \\ \Rightarrow (\text{Extend}(\text{Unconstrained Scope}, s)) \end{aligned}$$

Equation 4-11. The unconstrained scope is the bottom of the scope lattice

Whether any other extension relationships are implied depends on the interpretation of scopes' theme sets – and there are three competing ones. To demon-

strate how they work, imagine an agent that has a selected set of themes and wants to know which assertions are valid. For example, in the sample topic map above, if we have selected *Nigel* as our point of view, which assertions are valid? Clearly, *Nigel's* statement about *Piotr* applies, but what about his statements about the *Thesis*?

- The “all themes” interpretation says that only scopes that have all of their constraining themes in the selected set are applicable. This would put the *Nigel BeforeThesis* and *Nigel AfterThesis* scopes out of bounds, invalidating their assertions based on the selected point of view.
- The “any theme” interpretation says that every scope that has at least one constraining theme in the selected set is applicable, and its assertions valid. Under this interpretation – which makes more sense in this case – all three of *Nigel's* statements are considered valid.
- The “no relation” interpretation holds that there are no default relationships between scopes, and an agent can impose any interpretation it wants. This puts more power in the hands of agents, but leaves topic map authors with no clear scope semantics, undermining consistent use of the mechanism.

All interpretations have their proponents and produce sensible results in different situations. Each of them also imposes a different extension lattice structure on the scopes model. Equation 4-12 gives formal models for the first two interpretations, while Figure 4-30 shows the results of applying them to some simple scopes, labelled with their constraining themes. (Extension is transitive, so redundant relationships were eliminated to clarify the diagrams.)

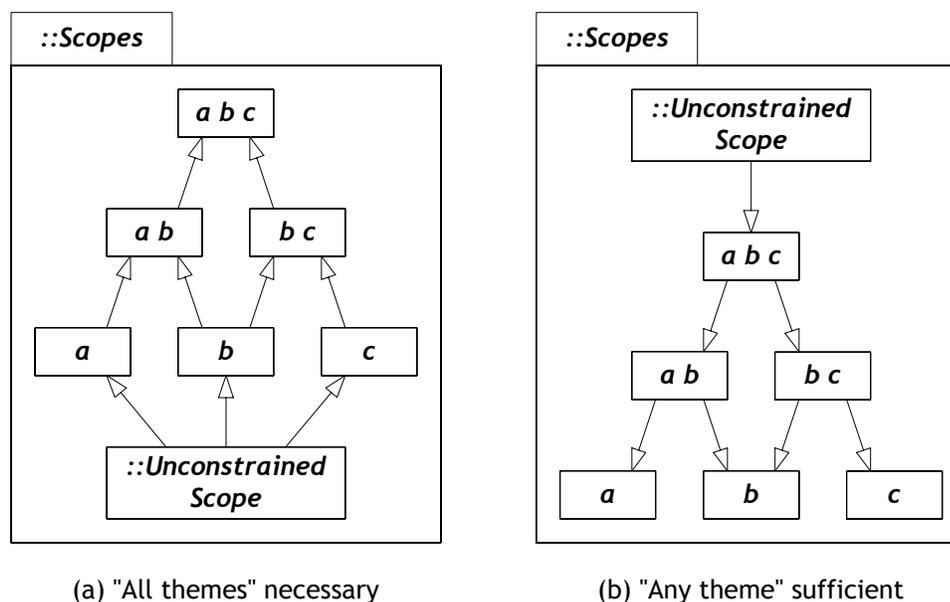
$$\forall s_1, s_2 : (s_1 \in \text{Scopes} \wedge s_2 \in \text{Scopes}) \Rightarrow \left(\forall t : \left(\begin{array}{l} \text{Constrain}(t, s_1) \\ \Rightarrow \text{Constrain}(t, s_2) \end{array} \right) \right) \Leftrightarrow \left(\text{Extend}(s_1, s_2) \right)$$

(a) "All themes" interpretation

$$\forall s_1, s_2 : (s_1 \in \text{Scopes} \wedge s_2 \in \text{Scopes}) \Rightarrow \left(\forall t : \left(\begin{array}{l} \text{Constrain}(t, s_1) \\ \Rightarrow \text{Constrain}(t, s_2) \end{array} \right) \right) \Leftrightarrow \left(\text{Extend}(s_2, s_1) \right)$$

(b) "Any themes" interpretation

Equation 4-12. Two formal interpretations of theme-based scope extension



(a) "All themes" necessary

(b) "Any theme" sufficient

Figure 4-30. Sample results of two interpretations of scope extension

With these lattices, it is much simpler for an agent to gather valid assertions based on a set of selected themes. In the "all themes" interpretation, an agent merely has to look in the scope constrained by all selected themes: assertions from subsumed scopes are already merged in. In the "any themes" interpretation, an agent has to create a new scope with no themes that is extended by all the others single-themed scopes matching the selected themes. This more complex procedure is necessary because, under this interpretation, applying a constrained scope means that *one or more* of its themes are selected, whereas an agent wants *all* selected themes to be in force. Another interesting consequence of the "any themes" interpretation is that the *Unconstrained Scope* inherits the themes of all the other scopes.

Since neither interpretation is appropriate in all situations, it seems likely that Topic Maps will eventually gain a more powerful scoping facility [PG01] that gives authors more control over the scope extension lattice.

4.3.5. Names in Topic Maps

Topic Maps deploy a flexible system for assigning names to subjects. Each subject can have any number of base names, which must be strings. Each base name may further have any number of variants of any nature. The variants could be translations to different languages, alternative strings to be used when sorting, visual or auditory renditions of the name, or any other resource that somehow represents the name. Both base names and their variants can be independently scoped.

LTM allows one to easily specify base names, but provides limited support for specifying variants. Only the two basic variants defined in the standard ontology are supported, and both must be strings: “sort names” (to be used when sorting the subjects) and “display names”.³⁵ Figure 4-31 gives *Piotr* the base name “Piotr Kaminski”, and *TheMatrix* the base name “The Matrix” with the sort name variant “Matrix, The”. Notice that this is one of the places where the subject (in this case, the name) is specified verbatim rather than by reference.

```
[Piotr = "Piotr Kaminski"]
[TheMatrix = "The Matrix" ; "Matrix, The"]
```

Figure 4-31. Assigning base and variant names to subjects

Since names are a Topic Maps primitive, there are no standard association types in the ontology, so we will reuse the NUO’s indication hierarchy in the mapping. Each base name introduces a new (abstract) name for the subject and a corresponding string representation. Each variant provides an alternative representa-

³⁵ In old versions of the Topic Maps standard, the base name was considered a unique trait of the subject, and two topics with the same base name for their subjects would be merged. Thus, a separate display name was required if the author wanted two topics to show the same name but not get merged. This “topic naming constraint” may not be retained in future revisions, in which case the “display name” variant would likely be dropped.

tion for the abstract name, scoped by the appropriate variant themes. (Non-string variants, though unsupported in LTM, are handled the same way.) Figure 4-32 shows the mapping of the topic map above to Braque.

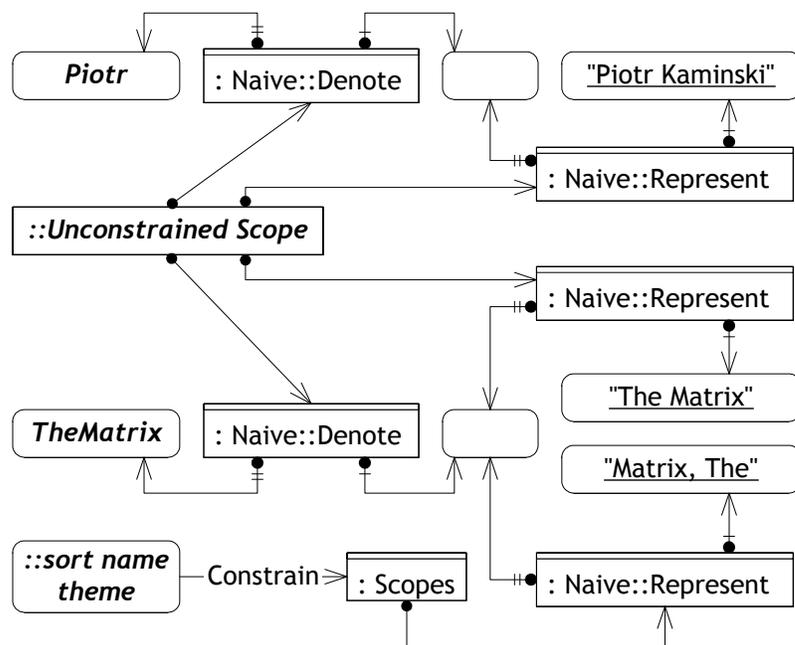


Figure 4-32. Mapping Topic Maps names to Braque

Arbitrary scopes specified by the author are modeled in the same way as for associations. A scope on a base name applies to both the *Denote* and the *Represent* relationships. A scope on a variant applies only to the new *Represent* relationship it introduces. The only additional Topic Maps restriction is that the themes of any variant scopes must be a superset of the base name scope's themes. Presumably, the intent is that whenever the base name is applicable, all the variant names should be applicable as well—that is, variant scopes should extend the base name scope. However, this effect is only achieved under the “any theme” interpretation; under the “all themes” interpretation, this restriction would result in variant scopes being more general than the base name scope.

4.3.6. Occurrences

Occurrences are the last notable feature of Topic Maps, and the one that ties them most strongly to their indexing roots. In a book's index, each entry lists the page

numbers where the topic is discussed: these are pointers to *occurrences* of the topic. While these occurrences could easily be described using associations, the designers of Topic Maps apparently felt that this feature was sufficiently important to rate its own primitive mechanism.

An occurrence is a binary relationship between a subject and a resource. The resource may be identified with an address, or entered in-line as a string. The occurrence may also have a type that more closely specifies how the resource is related to the subject, and a scope to constrain its validity. Figure 4-33 shows two sample occurrences written in LTM. The first gives the address of a resource related to the subject, and the second provides an in-line description of its subject, scoped by the *English* language. (Note that for some reason LTM requires all occurrences to have an explicit type.)

```
{ Braque, specification, "http://www.ideaest.com/research/documents/Thesis.doc" }
{ Braque, description, [[Peter's pomset-based metamodel]] } / English
```

Figure 4-33. Sample occurrences

Occurrences are very close in both structure and meaning to RDF triples, so we will model them the same way. Occurrence types are mapped to binary relations, and individual occurrences to member relationship ordered pairs. The *Occurrences* relation contains all occurrences of unspecified type, and is the superclass of all occurrence relations. Figure 4-34 shows the relation and assorted roles.

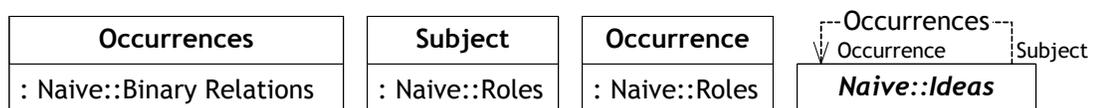


Figure 4-34. Occurrences relation and roles

Figure 4-35 shows the mapping of the sample occurrences above into Braque.

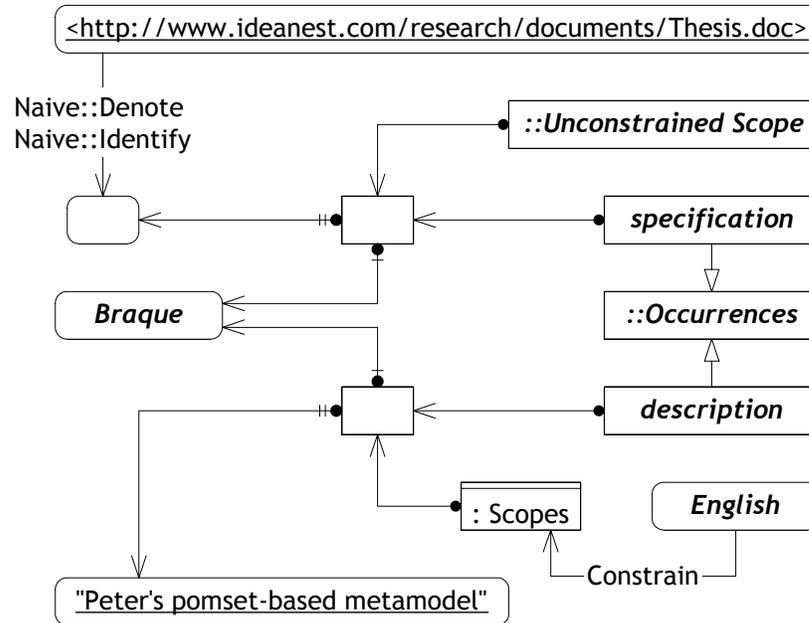


Figure 4-35. Sample mapping of occurrences into Braque

4.4. Integration Example

The previous sections discussed the metamodel mappings in detail; the time has come to put all this material together. This section presents a small example of integration, amalgamating three models, one of each kind (XML, RDF and Topic Maps). The goal is to show that models of different kinds can be integrated into one structure at the same semantic level using a generic mapping.

The plot behind the example is as follows. The University of Victoria is creating a new Bachelor of Software Engineering degree program (see <http://www.cs.uvic.ca/~hausi/sedp/>). The list of courses offered as part of the program has been encoded in RDF (Section 4.4.1), so that it can easily be merged into the university's course calendar. To attract students to the new program, the director has decided to show how the courses are relevant to the technical job market. She has located a number of job ads on the web, encoded in XHTML 2.0 [MA+02], an XML vocabulary (Section 4.4.2). She then asked the faculty and the employers for help in rating the relevance of material taught in the various courses to the job positions. Since many of the courses have not yet been taught even once, and

not everybody agrees on the exact content of each course, the ratings are somewhat subjective. In order to properly attribute the opinions, she has modelled the results as a topic map, with extensive use of scoping by authorship (Section 4.4.3). Finally, all three models are merged in Braque to provide students with a unified view of the information (Section 4.4.4).

Although the example is realistically motivated, some of the details are of necessity a little contrived to keep the demonstration focused. Only representative fragments of the models are presented, to keep the size manageable. The models are carefully set up to work together, since large-scale ontology alignment is outside the scope of this thesis. Finally, while the models try to showcase the unique qualities of each metamodel, they do not exercise every possible feature—for instance, it is difficult to find a practical use for RDF reification.

4.4.1. Course List in RDF

With the University of Victoria being at the forefront of semantic web development, the program committee has encoded the list of new courses in RDF. Although the abbreviated serialization syntaxes have a higher information density, and hence would be more appropriate for large data sets, fragments of the model are reproduced here in NTriples for simplicity. To keep the documents readable, the serialization substitutes short prefixes for the invariant part of each URI (Table 4-1). Prefixes are not part of the standard, but to obtain a valid NTriples document simply replace each prefix (including the colon) with its matching URI, concatenating it with the given suffix, and surround the whole construct with angled brackets.

Prefix	URI
rdf:	http://www.w3.org/1999/02/22-rdf-syntax-ns#
rdfs:	http://www.w3.org/2000/01/rdf-schema#
bseng:	http://www.engr.uvic.ca/bseng/courses#

Table 4-1. Prefix substitutions for RDF course list

With these prefixes in place, Figure 4-36 presents some fragments of the RDF course list. The first block introduces the model's schema, relating it to standard RDF classes and properties. The next two blocks give some information about two of the courses offered in the program. The prerequisite courses referenced are assumed to be defined elsewhere in the model, but are not shown in this fragment. Parts of the syllabi are also elided to conserve space. More properties and more courses could easily be added, but would not contribute to this example.

```

bseng:Course    rdf:type          rdfs:Class .
bseng:code      rdf:type          rdf:Property .
bseng:code      rdfs:subPropertyOf rdfs:label .
bseng:prereq    rdf:type          rdf:Property .
bseng:syllabus  rdf:type          rdf:Property .

bseng:IKM       rdf:type          bseng:Course .
bseng:IKM       bseng:code        "IKM" .
bseng:IKM       rdfs:label        "Information and Knowledge Management" .
bseng:IKM       bseng:prereq      bseng:DB .
bseng:IKM       bseng:syllabus    _:IKMsyl .
_:IKMsyl        rdf:type          rdf:Seq .
_:IKMsyl        rdf:_1            "Information models and systems: History and..." .
_:IKMsyl        rdf:_2            "Database systems: History and motivation for..." .
...syllabus continues...
_:IKMsyl        rdf:_12           "Privacy and civil liberties: Ethical and..." .

bseng:HCI       rdf:type          bseng:Course .
bseng:HCI       bseng:code        "HCI" .
bseng:HCI       rdfs:label        "Human-Computer Interaction" .
bseng:HCI       bseng:prereq      bseng:SE3 .
bseng:HCI       bseng:prereq      bseng:WE .
bseng:HCI       bseng:syllabus    _:HCIsyl .
_:HCIsyl        rdf:type          rdf:Seq .
_:HCIsyl        rdf:_1            "Foundations of human-computer interaction..." .
...syllabus continues...
_:HCIsyl        rdf:_7            "HCI aspects of collaboration and..." .

```

Figure 4-36. RDF course list fragment

Figure 4-37, Figure 4-38 and Figure 4-39 show the mapping of this RDF fragment to Braque.

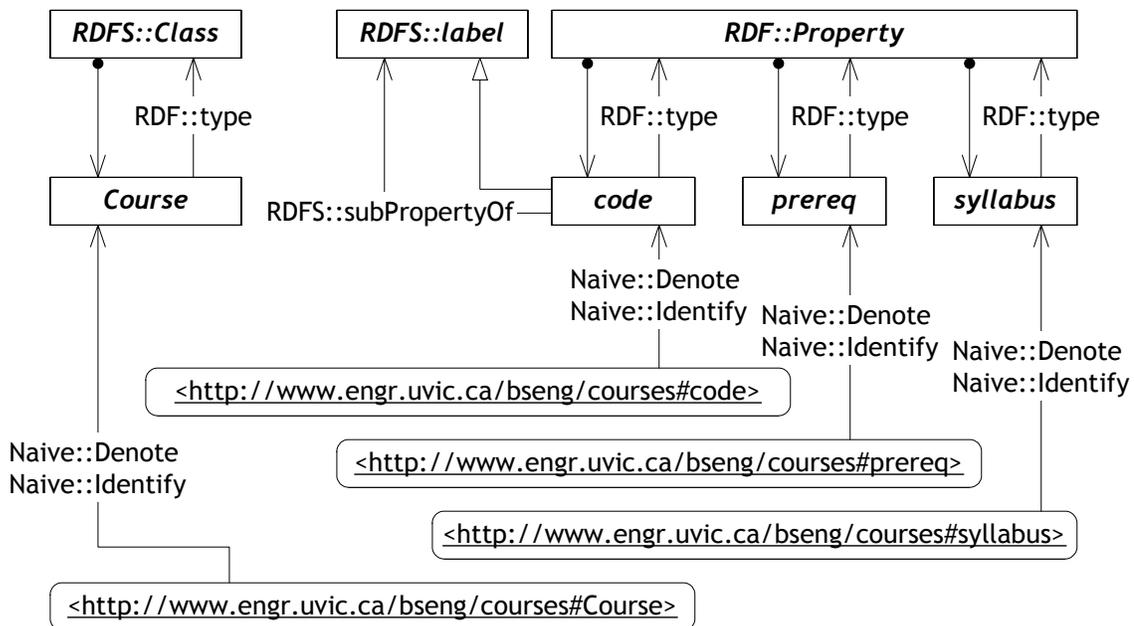


Figure 4-37. Course list schema in Braque

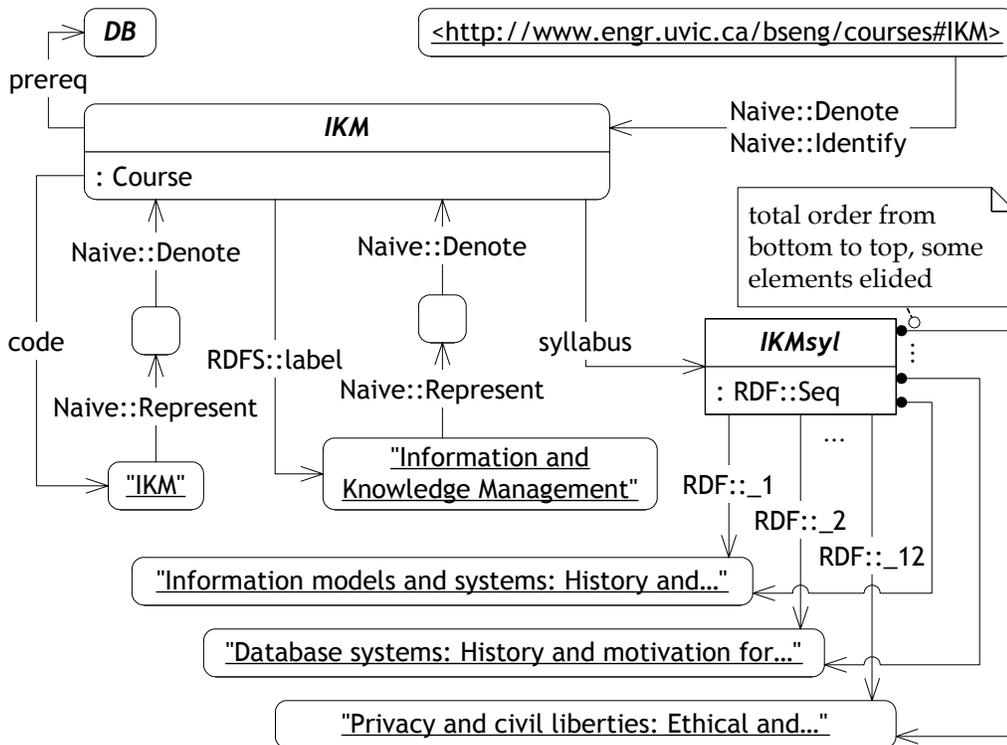


Figure 4-38. IKM course in Braque

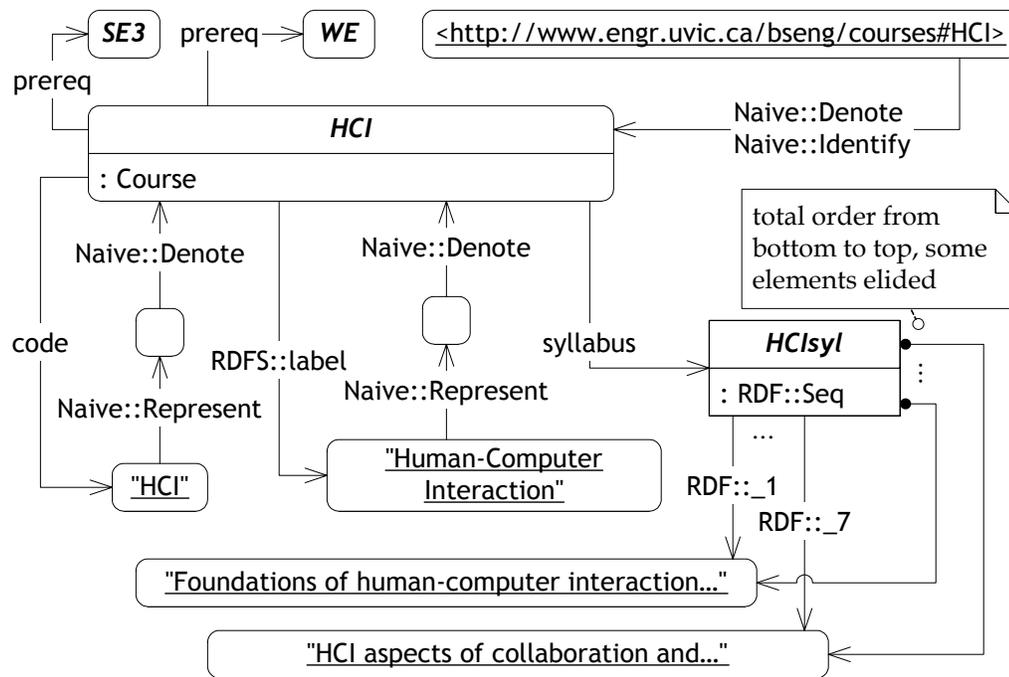


Figure 4-39. HCI course in Braque

4.4.2. Job Offerings in XML

High-tech companies, being at the forefront of technology, post open positions on their web sites, encoded in XHTML 2.0 [MA+02]. Since XHTML is an XML vocabulary, the documents can be imported directly into Braque. Figure 4-40 shows sample postings for two positions at the fictitious IdeaNest Inc. Note that the positions are embedded within anchor (“<a>”) elements with IDs, to make referencing them simpler. Had they been missing, we would have had to resort to structural XPath [CD99] expressions to identify relevant document fragments. The document is located at <http://www.ideanest.com/jobs.html>; this will help us derive the full URIs that correspond to the IDs.

```

<?xml version="1.0"?>
<!DOCTYPE html>
<html
  xmlns="http://www.w3.org/2002/06/xhtml2"
  xmlns:html="http://www.w3.org/2002/06/xhtml2"
>
  <head>
    <title>IdeaNest Entry-Level Positions</title>
  </head>
  <body>
    <a html:id="job425">
      <h>Software Engineer</h>
      <p>
        You will be programming in Java, implementing high-level designs. You will
        apply design patterns in the small to produce elegant, maintainable, well-tested
        code. Interest in knowledge management a big plus.
      </p>
    </a>
    <a html:id="job478">
      <h>Visualization Developer</h>
      <p>
        You will come up with novel ways of visualizing complex information. You will
        prototype your designs and conduct user studies to analyze and iteratively
        improve your prototypes.
      </p>
    </a>
  </body>
</html>

```

Figure 4-40. XHTML job offerings example

Figure 4-41 shows the Braque mapping of the XHTML fragment's namespace, element types and attribute types. Figure 4-42 show the mapping of the model's structure, augmented with the vocabulary-specific inference of Equation 4-13. The equation looks complex, but simply says that every HTML element with an "`<id>`" attribute has a URI that is the concatenation of the base document's URI and the attribute's value. When an XML document is read in, all its elements and attribute relationships are put into an *XML::Documents* nest, which then represents the resource identified by the document's URI.

$$\begin{aligned}
 & \left(\begin{array}{l}
 n \in \text{XML} :: \text{Top Namespaces} \wedge \\
 \text{Naive} :: \text{Denote}(\langle \text{http} : // \text{www.w3.org/2002/06/xhtml2} \rangle, n) \wedge \\
 \text{Naive} :: \text{Identify}(\langle \text{http} : // \text{www.w3.org/2002/06/xhtml2} \rangle, n) \wedge \\
 \forall a, n, i, d, r, b : \\
 i \in n \wedge \text{XML} :: \text{Local part}(i, \text{"id"}) \wedge \\
 i \in \text{XML} :: \text{Qualified Names} \wedge i \in \text{XML} :: \text{Attribute Names} \wedge \\
 a \in i \wedge a \in d \wedge d \in \text{XML} :: \text{Documents} \wedge \\
 \text{Naive} :: \text{Represent}(d, r) \wedge b \in \text{Domains} :: \text{URIs} \wedge \\
 \text{Naive} :: \text{Denote}(b, r) \wedge \text{Naive} :: \text{Identify}(b, r)
 \end{array} \right) \\
 \Rightarrow & (\text{Naive} :: \text{Denote}(b + a[2], a[1]) \wedge \text{Naive} :: \text{Identify}(b + a[2], a[1]))
 \end{aligned}$$

Equation 4-13. XHTML identification inference

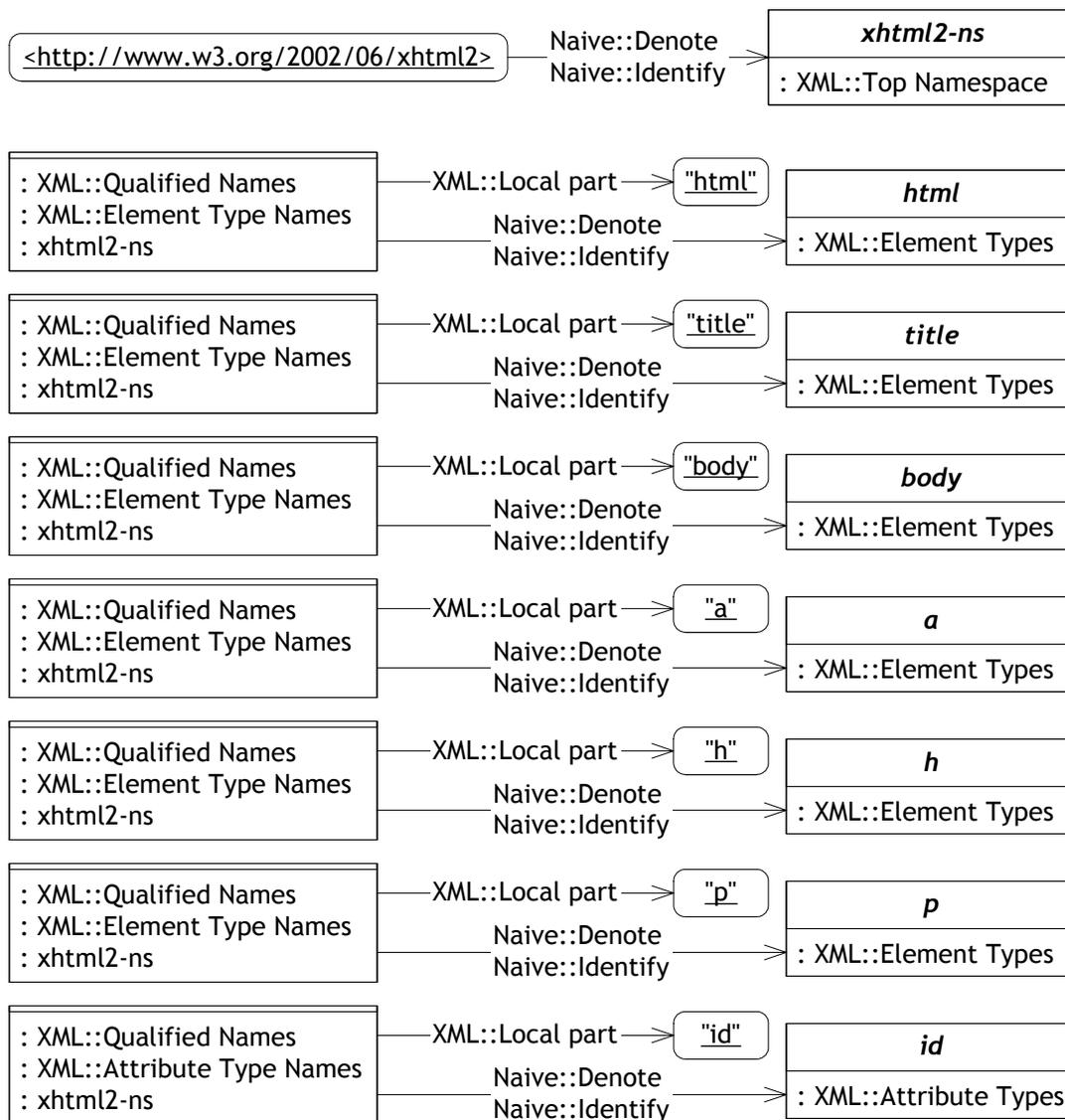


Figure 4-41. XHTML schema fragment in Braque

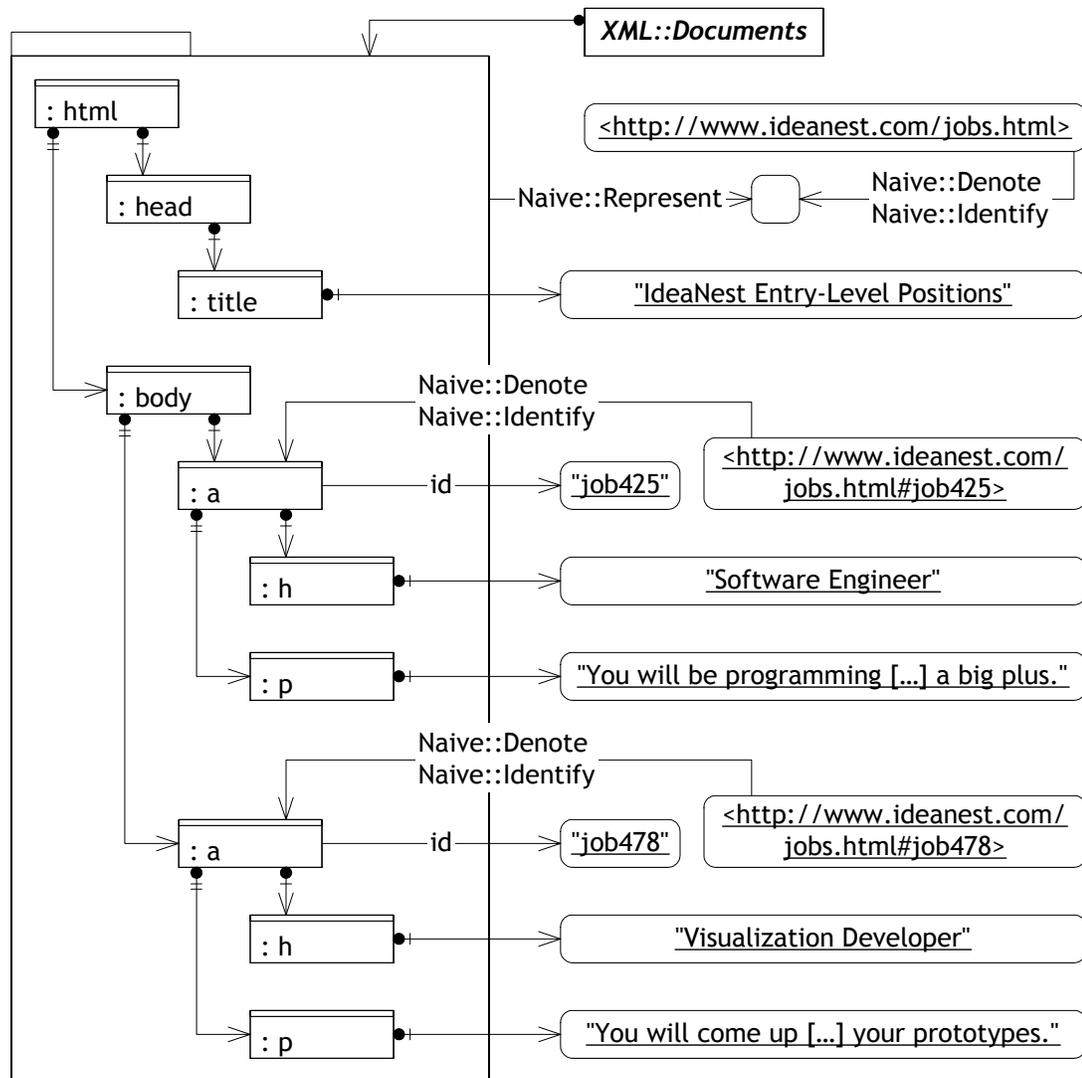


Figure 4-42. XHTML model fragment in Braque

4.4.3. Coverage Opinions as Topic Maps

A few days after the director puts out the call for help with relevance ratings, opinions start coming in. Since the director did not have time to build an interactive web page to collect the data, most of the answers come in email messages and she decides to create a topic map to integrate them. 3-ary associations can accurately record opinions,³⁶ scoping makes it easy to track their authors, and the

³⁶ Another option would have been to record binary job-course associations, then reify them to assign a relevance rating with another binary association. The decision whether to use n -ary associations or to reify and use binary ones is often arbitrary.

LTM syntax is better suited to manual input than NTriples or any XML vocabulary.

Before the director can begin entering the opinions arriving in her mailbox, she must articulate her resource identification strategy. She decides to use email addresses as subject identifiers for people, since they are reasonably unique and stable, and easy to pick out from the incoming messages. She also decides to use the job offerings' anchor IDs as subject identifiers; it would be wrong to use them as subject addresses for the jobs, since the URIs identify HTML fragments, not abstract job positions. On the other hand, the RDF course list is using the course URIs to directly identify the courses, so they must become subject addresses in the topic map for the integration to work properly.

The last problem facing the director is that she asked respondents to assign relevance ratings on a scale of 1 to 5 (1 being least relevant), but Topic Maps does not allow literals to participate in associations. Luckily, she remembers a proposal for assigning standard URIs to common data values [Sti01], so she creates five topics with subjects addressed by the URIs for integer values 1 through 5.

Figure 4-43 presents a fragment of the topic map containing a handful of topic declarations and opinion entries.

```
[ se_job  @"http://www.ideanest.com/jobs.html#job425" ]
[ vis_job  @"http://www.ideanest.com/jobs.html#job478" ]

[ IKM %"http://www.engr.uvic.ca/bseng/courses#IKM" ]
[ HCI %"http://www.engr.uvic.ca/bseng/courses#HCI" ]

[ piotr = "Piotr Kaminski"  @"mailto:piotr@ideanest.com" ]
[ jdoe = "John Doe"        @"mailto:jdoe@uvic.ca"
  @"mailto:john@cs.uvic.ca" ]

[ 1 %"x:int:1" ] [ 2 %"x:int:2" ] [ 3 %"x:int:3" ] [ 4 %"x:int:4" ] [ 5 %"x:int:5" ]

coverage ( se_job : job, IKM : course, 4 : relevance ) / piotr
coverage ( vis_job : job, IKM : course, 4 : relevance ) / piotr
coverage ( se_job : job, IKM : course, 2 : relevance ) / jdoe
coverage ( vis_job : job, HCI : course, 5 : relevance ) / jdoe
```

Figure 4-43. Coverage opinions topic map

Figure 4-44 shows the mapping of the topic definitions into Braque, while Figure 4-45 show the associations, roles and scopes. Note that mapping the “x:int” URIs to the numbers they denote requires a synthetic relation of infinite cardinality, of which only the relevant members are shown here.

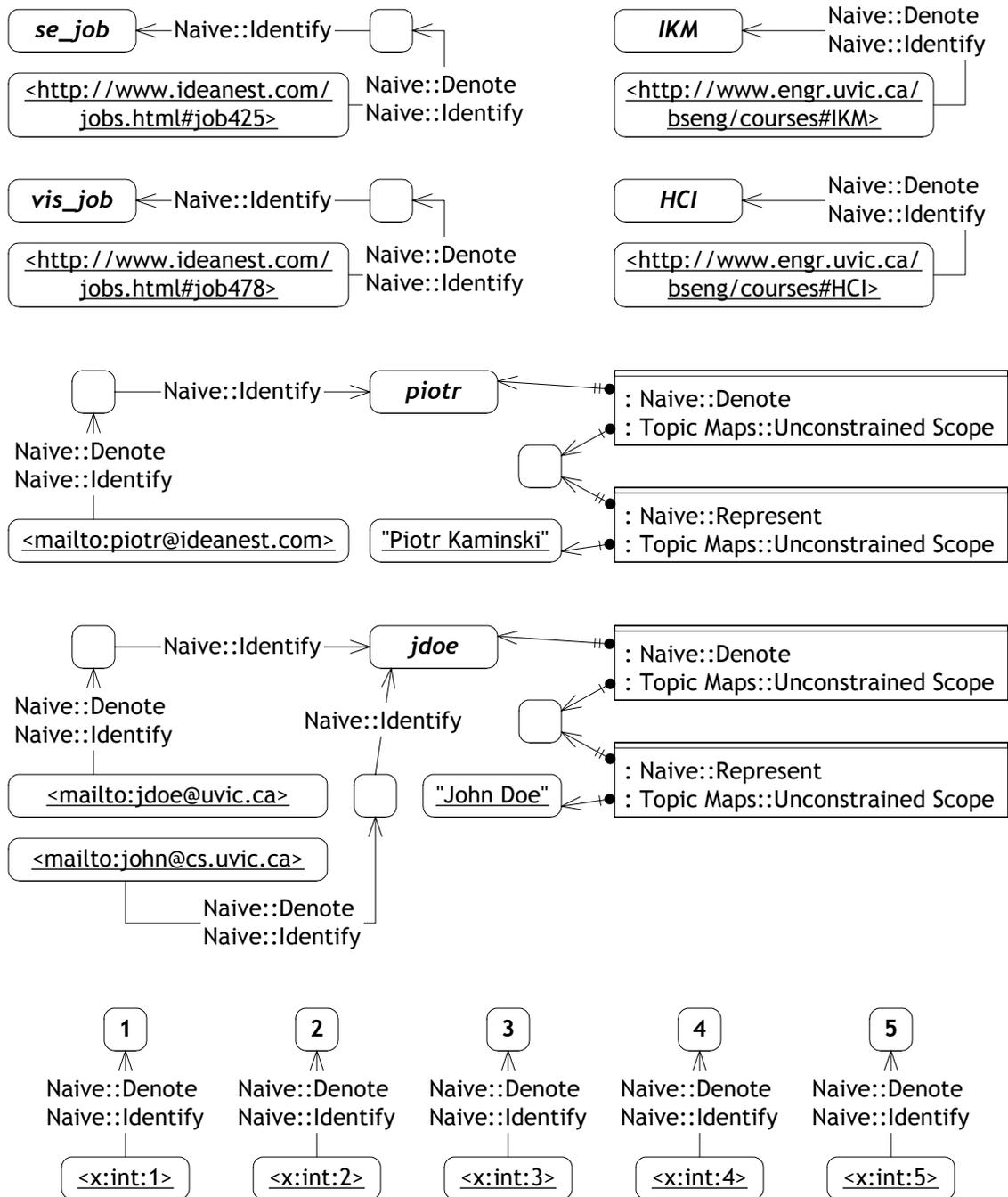


Figure 4-44. Topic map definitions in Braque

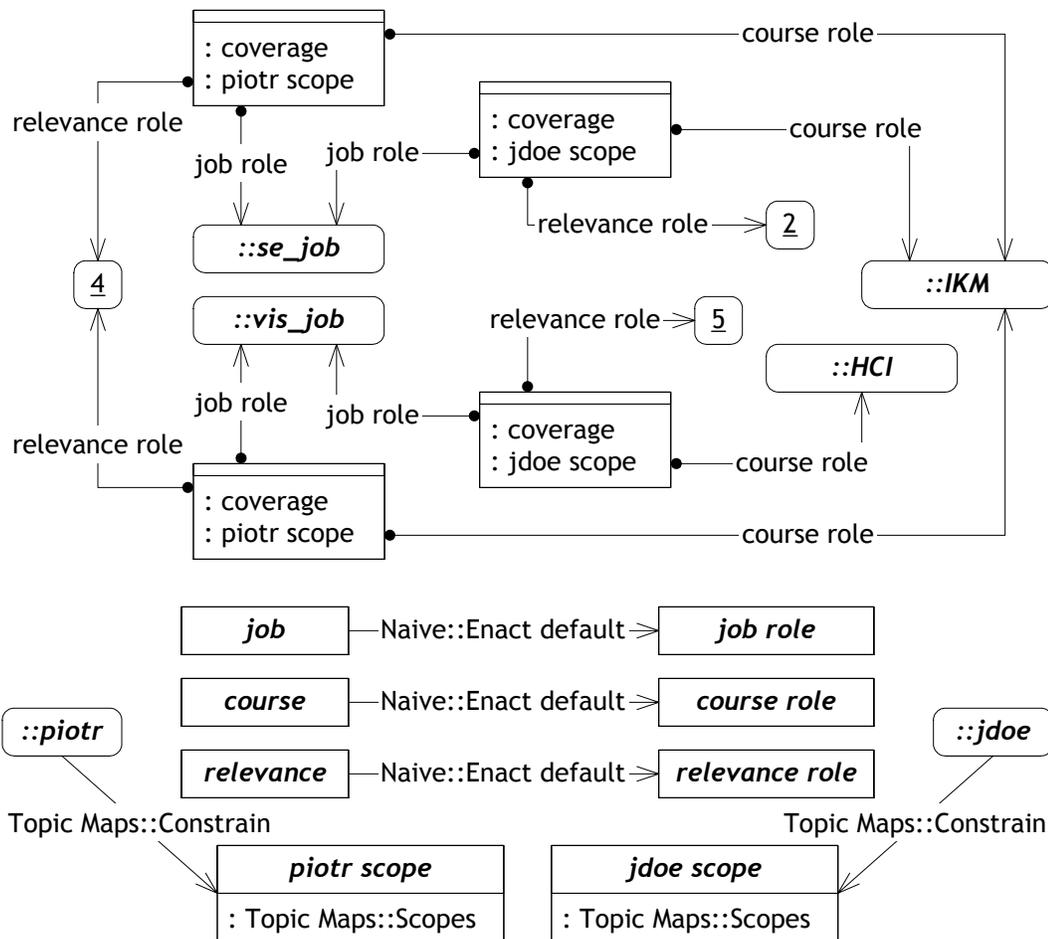


Figure 4-45. Topic map associations in Braque

4.4.4. Integrated Model in Braque

With all three models mapped into Braque, integrating them is just a question of merging the ideas they have in common. This process could be done by hand, but in this example the URIs were carefully chosen to automatically “line up” the models so it is sufficient to globally apply the “Identify is a function” rule (Equation 3-5). The appropriate job and course ideas are merged, connecting the models. Figure 4-46 shows only the essentials of the integrated model to cut down on the clutter, but all the ideas and relationships presented in previous diagrams are still there.

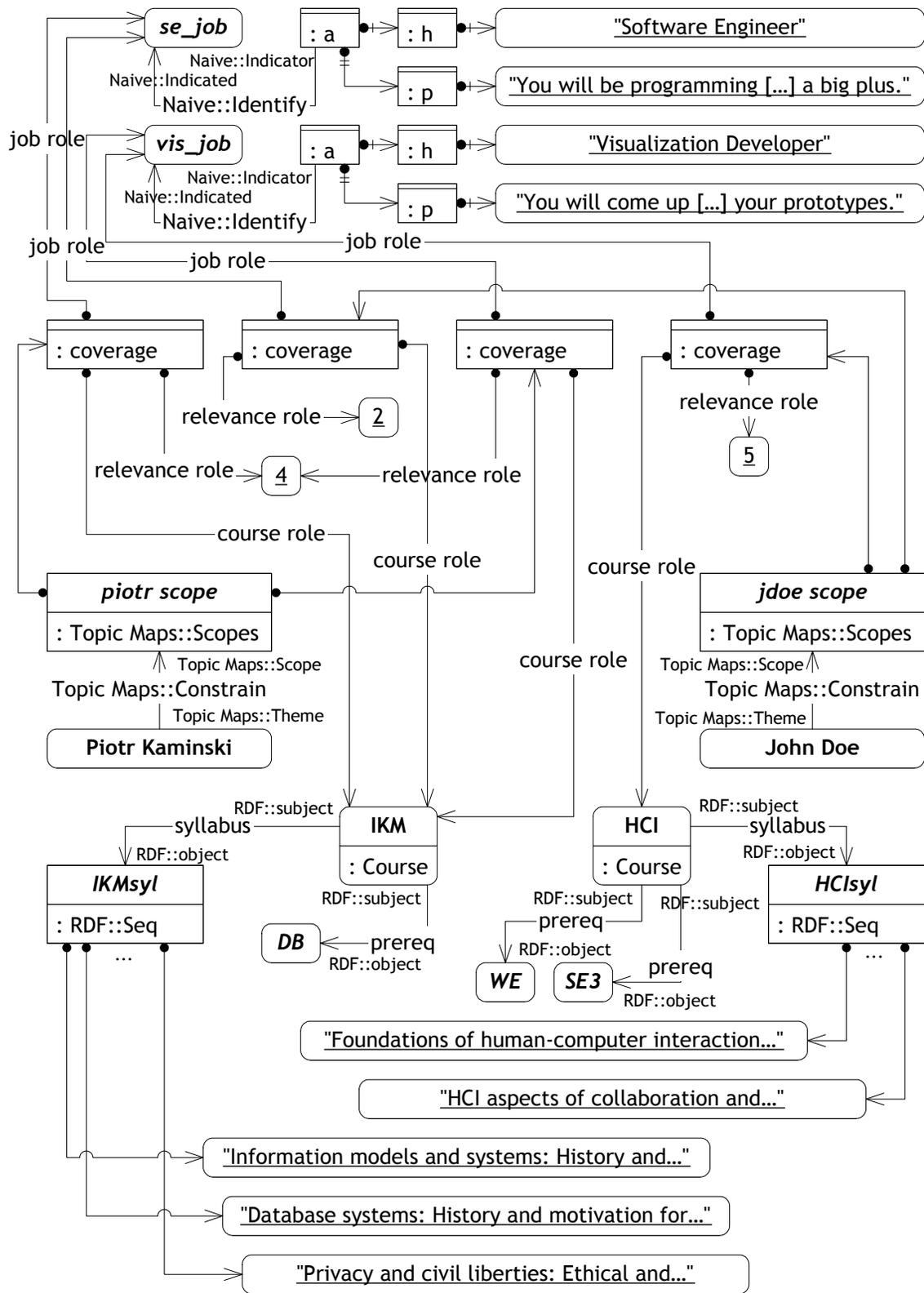


Figure 4-46. Essentials of the integrated model in Braque

With the models integrated, a student could now browse the employment offers, navigate seamlessly to the relevant course descriptions, and examine the courses' prereqs to help build his study schedule, or perhaps navigate back to all the ads relevant to a course to see what other career paths it enables. The director has a full record of the data collected, not just averages; if she wanted to, she could even attach the coverage associations to the email messages they were derived from. All information is preserved, and the three models could be extracted from the union and exported back in their original formats.

Moreover, not only are the models integrated structurally, but the RDF and Topic Maps models are also integrated semantically. Figure 4-46 displays all the inferred roles on the binary relationships (Section 3.3.4 and Equation 3-8), so both RDF and Topic Maps relationships' members can be uniformly accessed by role. An agent navigating or querying the integrated model does not need to know the original metamodel of each fragment. (Uniform access by index is possible too, if appropriate *Play Role by Index* declarations are set up for the Topic Maps association types.)

Even better integration is possible, of course, but requires manual intervention at the level of specific ontologies (semantic level). For example, the RDF model's *Course* class and the topic map's *course* type should probably be merged into one classifier, and some better semantics imposed on the XML model. However, automated object level integration provides a useful basis for further ontology mapping, which is outside the scope of this thesis.

5. Related Work

Modelling and integration is a crowded field, with multifarious approaches vying for dominance. This penultimate chapter take a critical look at the salient tools and ideas to put Braque into perspective.

5.1. The Integrated Metamodels

This section takes another, more critical look at the metamodels that were integrated in Chapter 4. Since the metamodels' structures were already explored in detail, this section merely critiques the metamodels' features in the context of the IMI Reference Model's object layer (Section 2.3.2) and the goals set out in Section 3.1, and demonstrates that none is expressive enough to comfortably integrate the other two.

5.1.1. Extensible Markup Language

XML [BPS00] by itself is a rudimentary metamodel, limited to a strict tree structure and unable to further describe the types of its elements. It gets more interesting when augmented with precise intra-document addressing (e.g., XPath [CD99] and XPointer [DMD01]), cross-branch linking constructs (e.g., XLink [DMO01]), and type specifications (e.g., XML Schema [TB+01][BM01] or RELAX NG [CM01]). With these additions, XML becomes a credible metamodel, and a possible semantic web dark horse.

The XML metamodel, however, is fundamentally different from other semantic web metamodels: it is not based on the assumption that a model reifies a world that consists of things and relationships. To put it another way, XML is about structure rather than meaning—about data, not information. That is not to say that XML *cannot* be used to express information, but rather that the specifications listed above do not take this point of view into consideration, and hence there is no standard interpretation for the data. It is thus meaningless (or at least premature) to talk about integrating other metamodels into XML.

Could a generic interpretation of augmented XML be constructed? Would such an interpretation be perceived as valuable by the XML community? These are still open research questions; some preliminary answers are explored in Section 5.2. Since XML and most of the extensions listed above have already achieved considerable traction on the Internet—much more so than all other semantic web metamodels combined—finding a way to integrate the extant data into the semantic web should be a high priority.

5.1.2. Resource Description Framework

The basic ideas of RDF are simple, though some of the particulars were badly influenced by the original XML serialization syntax. Why can a literal not be the subject of an assertion? Why can an assertion's predicate not be a blank node? Why are all literals strings? These small issues are annoying but irrelevant in the long term, since they could all easily be fixed.

More important is RDF's implementation of the object layer features from the IMI Reference Model. Right from the start, RDF runs into trouble by partially conflating object identity with URI equality: each (non-blank) node is labelled with a single distinct URI. In theory, this is correct, since each URI is supposed to identify a single resource [BM+98a], but in practice, this assumption is not always warranted.³⁷ A metamodel that cannot deal gracefully with authors' conflicting assumptions will not succeed in integrating information on the semantic web. A simple way to fix this problem would be to eliminate URI-labelled nodes and replace them with blank nodes with an explicit relationship to their URI identifier literal, to allow RDF agents to control resource identification using existing metamodel mechanisms. Although this has been informally suggested be-

³⁷ Some claim that this unique identification is an axiom, and hence true by definition, and it is the *meaning* given to the resource that changes. There are many ways to dance around the issue, and much electronic ink has been spilled arguing about it, but in the end squeezing the balloon in one place just makes it bulge somewhere else.

fore [Haw02], it is very unlikely that such a controversial change could officially be made to RDF.

RDF works fine for the next two features—binary relationships and basic typing—but its handling of the last three leaves much to be desired. In RDF, statement reification is crucial for establishing the provenance of assertions, yet the mechanism is unbearably clumsy. Reifying all the statements in a model quintuples its size without introducing any new information, and there is no way to connect a reification to the original stating. As for ordering, the use of ordinal properties to indicate membership results in very weak semantics for containers (see [Hay02a] Section 3.2.2). OWL [DC+02] will probably introduce a linked-list-like container construct to remedy some of the semantic issues, but as any first-year Computer Science student knows, this linear data structure is not appropriate in all circumstances. Finally, perhaps the best evidence that RDF’s implementation of reification and containers is ineffective is that these mechanisms are rarely used “in the wild”. [MA+02] found no examples of use at all, while the raw data (<http://www.i-u.de/forum/rdf/text-with-tab-2002.zip>) associated with [Ebe02] shows some use of containers, but less than a dozen valid uses of reification in over 250000 assertions.

RDF also has no support for representing n -ary relationships (for $n > 2$): they have to be constructed from n binary relationships. These binary relationships are attached to a central node that represents the n -ary relationship as shown in Figure 5-1.

```
_:x <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> _:Teach
_:x _:Teacher "Piotr Kaminski".
_:x _:Course "SEng 330".
_:x _:Section "F01".
```

Figure 5-1. A ternary relationship in RDF

This mechanism has many problems, the most flagrant being that n -ary relationships are completely different from binary ones. The type of an n -ary relation-

ship is indicated with a *type* predicate, like for any normal resource, whereas as binary relationship's type *is* the predicate. The subject and object of a binary relationship are primitive features, indicated by their position in the triple, while each components of an n -ary relationship must be specified with a separate statement. These differences force n -ary relations to be treated differently when defining ontologies or querying RDF databases. N -ary relations are already rare, and this second-class treatment is likely to further encourage their replacement with inaccurate workarounds.

Overall, RDF is best suited to publishing simple information in documents where the meaning of URIs can be centrally controlled. Its inadequate implementation of object layer features makes it useless for model integration.

5.1.3. Topic Maps

The first criticism that might be levelled at Topic Maps is that the metamodel contains all kinds of constructs that do not necessarily apply to all models, such as occurrences, complex naming, and perhaps even scoping. The committee has recognized this shortcoming, and separated the specification into two layers: the base Reference Model [NB02] and the Standard Application Model [GM02] built upon it. Although the relationship between these two models has not yet been fully explicated, the remainder of this critique concerns itself only with the Reference Model.

The associations used by Topic Maps elegantly satisfy the binary and n -ary relationship requirements of the IMI object layer. Basic typing is well done for subjects, though the limitation of associations to a single type seems arbitrary. The reification facilities are also excellent, since they espouse pre-emptive reification [New02] and do not require the introduction of any extra assertions *à la* RDF.

Unfortunately, Topic Maps subject identification suffers from the same problems as RDF's: URIs are too closely coupled to topic nodes. Topic Maps also have no

built-in ordering mechanism, though it would be possible to introduce ordinal properties similar to RDF, with the same concomitant problems.

Overall, stripped of their extra baggage, Topic Maps are a decent generic meta-model, richer than RDF. The scoping and naming mechanisms could use some refinement, but are definitely worthy of being emulated in any high-level ontology. However, the strict subject identification cannot be scoped, which makes it difficult to integrate models that may not agree on the meaning of a URI. The lack of ordering also makes it very difficult to integrate any models where this is a primitive feature, such as XML.

5.2. Merging RDF and XML

Ever since RDF saw the light of day, some people have been trying to justify the need for a separate metamodel that used XML for syntax but ignored its structure [Ber98][ST99]. Others have been trying to merge the models in various ways. Of course, the problem (as identified earlier in Section 4.1) is that it is impossible to ascribe semantics to arbitrary XML documents beyond their structure (defined in the XML Information Set [CT01]).

This section presents an overview and criticism of some draft techniques for unifying XML and RDF. Some of the approaches introduce a new metamodel. Though these metamodels were only meant to hold XML and RDF information, I nonetheless evaluate their potential for more wide-scale metamodel integration.

Note that none of the proposals attempt to unify XML qualified names and RDF URIs; only the last one (Section 5.2.3) even mentions it. Such unification is easier if names are explicitly represented in the model (as in Braque), but rather tricky when they are metamodel primitives.

5.2.1. Bridging the Gap

Melnik's proposal [Mel99] is the simplest: he suggests a technique for expressing arbitrary XML documents with an unmodified RDF metamodel. By default, both

attribute and element labels are considered to describe roles and are mapped to RDF properties (edge labels) in the model. Any character content mixed in with elements is mapped to literals and attached to the parent element with ordinal properties, treating it as a container. However, if there is only one child, it is attached to the element as an *rdf:value* property instead.

Since not all element labels name roles, Melnik also defines a mechanism to treat them as element type names instead. When interpreted as such, the element is given an *rdf:type* identified by its label, and is attached to its parent using an ordinal or *rdf:value* link, like the literals above.³⁸ This special treatment is triggered by attaching an *instance* attribute to the element. This attribute can be specified either in the XML document itself, or in the document's schema (a DTD [BPS00] in Melnik's paper, but XML Schema [TB+01] would work as well). A few other attributes provide finer control over the mapping.

Melnik's mapping is representative of what can be done to integrate XML data into RDF models. Some of the fine points could definitely be improved:

- Mapping details might be controlled through a third document instead of modifying the original or its schema.
- *rdf:value* ought to be abandoned to make the mapping more consistent.
- The mapping should differentiate between properties induced by elements and properties induced by attributes, perhaps by introducing new subclasses of *rdf:Property*.

Overall, though, one cannot do much better within the boundaries of the RDF metamodel. The ordering of children elements with role-like labels cannot be preserved, since the predicate can only express either the role or the ordinal.

³⁸ Apparently, the algorithm for every element is to first attach all children elements with role-like labels using each role's matching property, then attach all remaining children literals and elements with type-like labels using ordinal properties, or *rdf:value* if only one remains.

There is also no way to get rid of the ugly ordinal properties that RDF uses to represent order.

5.2.2. OrdLab Graphs

Boley realizes that force fitting an ordered nesting structure onto labelled property edges is a non-starter, and proposes a combined metamodel [Bol01] that allows mixing of ordered and labelled edges.

Boley starts with XML document trees, where each element is represented by a vertex labelled with the element's name as its type and with the element's contents as its substance. The element's substance label consists of its character content (call this a literal vertex), or remains empty if the element only contains element children and no text (call this a structure vertex). Mixed content—an element containing both text and children elements—is not allowed in the model. Structure vertices have directed edges going to their children, and each vertex's outgoing edges are totally ordered, reflecting the document order of markup. Element attributes are not represented in the tree.

On the RDF side, the graph is based on the usual definition [Bec02]. Each resource is a vertex labelled with its type and its URI (or with its literal, or left blank). A special “any” type is used in the graph if the resource lacks an RDF type, and special (but vaguely defined) intersection types are introduced if the resource has multiple RDF types. Unordered directed edges, labelled with property URIs, link pairs of resources.

The two metamodels have differences in edge ordering, edge labelling and label syntax. The unified metamodel is, in Boley's own words, the “least general generalization” that encompasses the two data models. Each vertex's collection of outgoing edges is composed of a list of totally ordered unlabelled edges and a set of unordered labelled edges. The unlabelled edges come from XML children elements and RDF *Seq* containers. The labelled edges come from RDF statements,

as well as from XML documents extended with role-playing indicators.³⁹ Boley also proposes some special element attributes that would allow vertices originating from XML and RDF to be merged based on their URI substance label.

Boley's metamodel is a good effort, but sadly incomplete. It does not completely model XML documents, missing attributes and mixed content, which are hardly arcane features. Holding the vertex type as a secondary label instead of making it a part of the graph structure is also a mistake, as demonstrated by the awkward mapping from RDF's multiple-classification type system. It should also be obvious by now that making labels out of resource URIs is not a good idea.

Edge ordering is without doubt the metamodel's best feature, but even this could be improved. The dichotomy between labelled and ordered edges is too restrictive: sometimes it is useful for an edge to have both features. The total order on the unlabelled edges should also be relaxed, to allow for mapping of RDF's *Bag* and *Alt* containers and so as not to force an arbitrary linear order when different vertices' edges are merged (e.g., when making inferences or otherwise manipulating the graph).

Since Boley's metamodel fails to completely subsume XML and RDF (its stated goal), it does not come as a surprise that it is inadequate for integrating metamodels in general. Taking Topic Maps as an example, there is no way to model associations in the graph without resorting to a lift (see Section 2.4.3) to represent the association itself as a vertex in the graph, breaking semantic continuity with the RDF and XML of the model.

5.2.3. The Yin/Yang Web

Patel-Schneider and Siméon have recently proposed a new "Yin/Yang" metamodel [PS02a][PS02b] that integrates the syntax of XML with the semantics of

³⁹ Initially, Boley proposes some fundamental changes to XML to support roles, but in the end gives reductions to some custom XML vocabularies similar to Melnik's attributes in the previous section.

RDF. Their approach is unusual, since they start from RDF's XML serialization (RDF/XML for short, not presented in this thesis) rather than its abstract meta-model and place great value in staying compatible with existing XML tools. Yin/Yang is also a rare example of a two-layer metamodel: the bottom data layer records the structure and data types of the XML document, while the more abstract top information layer records the semantics.⁴⁰ A model is always composed of both layers; the data layer is not discarded.

The bottom layer is the XQuery [BC+02A] data model [FMG02], which relies on the Post-Schema Validation Infoset of XML Schema [TB+01], which itself is built upon the XML Information Set [CT01]. This modeling stack has a steep learning curve, but allows the Yin/Yang metamodel to leverage the tools already developed for the incorporated standards, thus simplifying its implementation.

The top layer is a directed graph whose meaning is defined by a model theory that, while logically very precise, can be difficult to understand. Each vertex in the graph represents a resource and may be unlabelled or labelled with any number of XML qualified names (Section 4.1.2). Resource classes are mapped to their extension (instance set), but the mapping is "outside" the model and hence essentially equivalent to attaching type labels to vertices. Unlabelled directed edges connect the vertices and each vertex has a strict partial order for its outgoing edges.

Translation rules connect the bottom layer to the top layer, assigning an arbitrary meaning to any XML document. Each XML element and contiguous block of text is translated to a vertex (representing a resource), and connected with ordered⁴¹ edges to the vertices of its children, if any. Each element's name is mapped to a vertex (representing a resource class) as well, and the element's vertex added to

⁴⁰ These layers are related to the semantic layers introduced in Section 2.3.1. The Yin/Yang data layer could be inserted between the IMI syntax and object layers, while the Yin/Yang semantic layer seems equivalent to the IMI object layer.

⁴¹ A special *order* attribute controls whether the document order is significant and should be represented in the graph.

its extension. (Naturally, the type model is unstratified.) Attributes are translated in a similar manner: each attribute becomes a vertex that is connected to its value vertex with an edge. Element vertices are connected to their attribute vertices.

This translation is used for both XML and RDF/XML documents, with a few exceptions that take into account the meaning of certain reserved RDF attributes. (For example, an *rdf:resource* attribute can be used to make a lateral link to another resource by giving its URI, thus escaping the strict XML tree structure.) Due to RDF/XML's striped syntax [Bri02], this direct translation produces a semantic graph different from—though translatable to—the usual RDF graph model [Bec02]. Statements are mapped to vertices instead of edges, so that an assertion results in the creation of a new predicate vertex with an incoming edge from its subject vertex and an outgoing edge to its object vertex. The predicate vertex is also the subject of an *rdf:type* statement that links it to its property type vertex,⁴² differing from the RDF model where the predicate type is expressed using an edge label instead.

The Yin/Yang metamodel succeeds at integrating XML and basic RDF in a novel fashion. However, it fails to deal with RDF containers and reification, whose integration might prove troublesome since their semantics are not a direct translation from the data model. The metamodel also makes the by-now familiar wrong choice of putting identification and classification outside the model, though it improves on OrdLab by allowing multiple names and types for each resource. The partial edge order is also an improvement, and modeling statements with vertices is a major advantage, since it makes them referenceable by other statements (e.g., to assert provenance). Nonetheless, Yin/Yang is still not expressive

⁴² This approach causes an infinite regress, since the new statement needs another statement to link it to the *rdf:type* property, etc. The authors introduce a tricky loop where a vertex is both the subject and predicate of a statement to keep the model finite.

enough to integrate Topic Maps models without a lift, since there is no way to assign roles to edges – either by label or reference.

It is more difficult to evaluate the unusual two-layer architecture of the meta-model. The XQuery data model allows the use of many standard (and powerful) tools, and its familiarity to many XML developers may well increase the Yin/Yang metamodel's acceptance. On the other hand, coupling so closely to RDF's XML serialization is risky. If another standard serialization emerges – XML or otherwise – its documents will need to be translated into RDF/XML by another tool before they can be imported into Yin/Yang. In addition, when mutating a model, it might be difficult to keep the semantic layer graph bipartite⁴³ to allow later export back to RDF. Actually, it might be altogether impossible to mutate models directly at the semantic level, since the proposal does not include any way to translate changes back to the data level to keep the layers synchronized.

Overall, Yin/Yang is a gutsy and original attempt at a metamodel that integrates very tightly with XML and its many adjoining specifications. Unfortunately, this quality makes it useless at integrating metamodels that lack an XML serialization, or whose XML format is too far from its semantic interpretation. Even stripped of its data layer, the semantic metamodel keeps too many mechanisms on the outside and lacks the referenceability needed to be truly expressive.

5.3. Merging RDF and Topic Maps

Just like for RDF and XML, no sooner had the first Topic Maps standard solidified than people noticed its proximity to RDF and tried to reconcile or further differentiate the two [Pep00], according to their inclinations. Actually, most the work was done by members of the Topic Maps community; the majority of RDF supporters seem to consider Topic Maps as irrelevant or overly complex.

⁴³ To stay compatible with the RDF metamodel, statement vertices can only serve as predicates, or participate as subjects only in *rdf:type* statements.

This section explores the various integration proposals for RDF and Topic Maps. All of the proposals are expressed as mappings between the metamodels; no one has tried to introduce a new metamodel except Bowers and Delcambre, whose project has a wider scope and is discussed in Section 5.4.3. The mappings are interesting historically, but also because the later ones take completely different (and sometimes novel) approaches to the problem.

5.3.1. The Early Lifts

The very first mapping from Topic Maps to RDF is probably Vlist's syntactic translator [Vli01]. An XSLT [Cla99] transform maps directly from XTM to RDF documents, taking advantage of similarities in their syntax. Vlist provides an example of the translation, but does not reveal the exact style sheet he used (though it is based on his earlier XLink transforms [Vli00]), so the mapping is impossible to analyze in detail.

Moore made the first published attempt at integration [Moo01]. Starting with inaccurate UML class models of both RDF and Topic Maps, he suggests a lift from RDF to Topic Maps and vice-versa, providing the barest amount of detail needed to convince reader that such lifts are possible. After this purely academic exercise, he concedes that "modelling the model" will not bring about integration and proceeds to construct object-level metamodel mappings from RDF to Topic Maps and back. Both mappings are confusing and incomplete; Moore does not deal with Topic Maps scopes, occurrences, subject types or names, nor does he deal with RDF types, containers or reified statements. Furthermore, his RDF to Topic Maps mapping also requires changes to the Topic Maps metamodel, and the reverse mapping misunderstands the nature of RDF assertions. Overall, Moore's best contribution was to bring more attention to the topic of metamodel integration, and to define the difference between object level lifts and mappings.

The next attempt was made by Lacher and Decker [LD01]. By leveraging an existing Topic Maps graph formalization [NB01], they quickly achieve a complete if

awkward lift from Topic Maps into RDF, claiming that an object level mapping is bound to lose information. Though they demonstrate the integration by showing a query that spans Topic Map and RDF information, the example is just sleight-of-hand. The user must know the precise boundaries of the lifted topic map, and manually account for the semantic differences in the query. Thus, Lacher and Decker do not achieve true semantic integration of the two metamodels.

Ogievetsky makes the final attempt of this initial trio [Ogi01]. The mapping is a lift from Topic Maps to RDF, also based on [NB01]. Ogievetsky makes a good-faith effort to take advantage of as much of RDF's semantics as possible, trying to avoid a lift. He is quickly defeated when he attempts to map Topic Maps associations, since RDF only supports binary properties without roles. He must resort to modeling each association as a separate resource, connected to its members with statements that use the roles as predicates. This, and a few other inconsistencies, clearly makes the mapping a lift, with all the concomitant disadvantages. Ogievetsky demonstrates – perhaps without being aware of it – that an object level mapping from Topic Maps to RDF is impossible.

5.3.2. Two Semantic Mappings

After reviewing and evaluating the three previous proposals, Garshol [Gar01b] concludes that the mappings proposed thus far are inadequate. Based on his analysis of equivalent sample models in both RDF and topic map form, he realizes that an object level mapping from Topic Maps to RDF is impossible, since RDF's relationships are not sufficiently expressive. In passing, he suggests a lift less awkward than those previously proposed, but then quickly moves on to discussing a semantic level mapping. Since no single semantic level mapping would work for all models (see Section 2.4.4), Garshol proposes a rule-based mapping generator. Naturally, the user must manually specify topic map queries [Gar01a] and RDF triples to be output for each ontology, but this allows for a custom translation that can find the most natural RDF expression for any Topic

Map assertion. Unfortunately, there is still no way to work around RDF's lack of n -ary relationships, so some associations will have to be lifted.

Turning to the other direction, Garshol asserts that an object level mapping from RDF to Topic Maps is also impossible, paradoxically because Topic Maps' semantics are too rich! Indeed, while Topic Maps' structure is certainly flexible enough to integrate RDF models, the standard also specifies primitive mechanisms for constructs that would be ad-hoc in RDF (e.g., names, scoping). Hence, he argues, any mapping that does not take these constructs to their proper Topic Maps equivalents is flawed, since the resulting topic map will not express common features in a way that Topic Maps processors will understand. He then proposes a semantic level rule-based mapping generator similar to the one above, which allows a user to specify the exact Topic Maps equivalent of each RDF property.

Garshol is completely right on at least one count: information cannot be created out of thin air. However, that does not automatically imply that object level mappings from information-poor to information-rich metamodels are impossible or useless. Indeed, a well-designed expressive metamodel should degrade gracefully when insufficient information is supplied, and allow the meaning of ad-hoc constructs to be filled in after they have been integrated. The Topic Maps metamodel is not well suited in this regard; the advanced features are primitive and critical to the correct functioning of Topic Maps application, and hence must be translated immediately. Given these constraints, Garshol's semantic mapping generator does a good job, but it also reveals that Topic Maps are a bad target for metamodel integration.

5.3.3. Occurrences as Statements

Following up on his academic first attempt, Moore comes back with some practical suggestions [Moo02] that can be easily implemented and do not require changes to the metamodels. He concentrates on an object level mapping from

RDF to Topic Maps, augmented with a handful of semantic mappings of common ontologies. RDF statements are mapped to Topic Maps occurrences, the predicate property becomes the occurrence's type, and a little prestidigitation ensures that both literals and resources get sensible equivalents.

Addressing Garshol's criticisms of object level mappings,⁴⁴ Moore notes that some of Topic Maps' advanced features are often expressed using a small set of standard predicates in RDF. He therefore adds primitive translations for *rdf:type*, *rdfs:label*, and the Dublin Core [WK+99] properties *dc:title* and *dc:identifier*. While the effect of adding just four extra translations may seem small, keep in mind that these are terms from popular upper ontologies, so any more specific properties are likely to be specializations and will be translated as well. There is no translation that creates scopes, which is understandable since the RDF community has not yet come to an agreement on how to represent contexts.

Thus far, the mapping could be turned around easily to take topic maps to RDF models, but Moore has not yet dealt with the thorny topic of associations; he never does. He says nothing about creating associations from RDF statements, and vaguely suggests a rule-based scheme similar to Garshol's for creating RDF statements from associations. Hence, the end product of Moore's paper is a simple object level mapping of the RDF structure and a few common terms into a subset of Topic Maps that excludes associations. This mapping is quite similar to the RDF introduced in Section 4.2, when the latter's results are interpreted from a Topic Maps point of view.

5.3.4. The Syntactic Web

Robie proposes a structural integration of RDF and Topic Maps into XML [Rob01]. Though both the RDF and Topic Maps standards already define XML serializations, they are very flexible, allowing the same semantic structure to be written down in many different ways. This flexibility makes them difficult to

⁴⁴ Actually, Moore was apparently unaware of Garshol's paper since he does not reference it.

query directly, since any syntax-based query has to take into account all possible ways to serialize a fact. Robie solves this problem by proposing normal serialized forms for both RDF and Topic Maps; original documents are deserialized into the appropriate model then reemitted in the normal form before being queried. The normal forms are very regular and closely match the structure of the abstract models, which makes them reasonably easy to query with XQuery [BC+02a].

An important detail is that the models are not completed with logical entailments before being reemitted. In RDF, for example, this means that if in the original document *A* is an *rdfs:subClassOf B*, and *X rdf:type A*, the fact that *X rdf:type B* will not appear in the normalized document. To get around this problem, Robie introduces a passel of XQuery functions that perform the required entailments on the fly; for example, *instance-of-class* recursively computes the transitive closure of *rdfs:subClassOf*.

In the end, there is no semantic integration, only normalized but disjoint syntaxes for the two metamodels. Robie's effort also demonstrates the futility of trying to operate directly on the syntactic representation of a model, even one that matches the structure very closely: the extra functions wind up creating transient fragments of the abstract model anyway.

5.4. Other Metamodels

Many other metamodels have informed the design of Braque. Due to the large number of candidates, this section groups them into broad categories and gives only a few general comments for each group.

5.4.1. Directed Binary Graphs

Labelled directed graphs are the most popular way of modeling semistructured data. Some systems that use them are OEM [PGW95][ASB99] (Object Exchange Model), Rigi [Won98], SHriMP [Sto98] (Simple Hierarchical Multi-Perspective

views), GOOD [GP+94] (Graph-Oriented Object Database), the Associative Model of Data [Wil00], Conceptual Graphs [SD99], and, of course, RDF. All but the last two are just data models, with no intrinsic meaning associated with the structures, which makes them unusable for information integration. However, even gifted with a standard interpretation, these metamodels are not expressive or flexible enough to integrate information with ease.

The main problem with directed graph metamodels, as mentioned in the discussion of RDF (Section 5.1.2), is that edges cannot be referenced. To participate in assertions, edges must be manually lifted (reified) into vertices, all the while ensuring that the whole model is at the same level of reification to provide a semantically level ground for queries and navigation. To avoid the reification, some metamodels support attributed edges (and sometimes vertices). Attributes are labelled primitive values attached to an edge. This change merely pushes the problem back one step—attributes cannot be referenced; do they need second-order attributes?—while making the metamodel more complex.

Directed graphs also do not support n -ary ($n > 2$) relationships, forcing the use of reification to obtain this feature. Other common issues include unreferenceable human-readable labels for which it is not possible to provide further machine-processable information, and a lack of ordering features.

Simple directed graph metamodels are usually sufficient if ontologies and models are created from scratch, but are not up to the task of integrating information.

5.4.2. Advanced Graphs

In response to the limitations of binary directed graphs, many researchers have started developing other varieties of graph metamodels (e.g., TGraphs [EF94]). The last few years have seen a number of movements towards an all-encompassing, universal graph metamodel that could be used in all domains. Some contenders include GraphML [BE+01], GXL [Win01] (the Graph Exchange Language), and XGMML [PK01] (Extensible Graph Markup and Modeling Lan-

guage). Although the details differ, the idea behind all of these proposals is to extend the metamodel to handle all the types of graphs used in mathematics: graphs with directed, undirected and mixed edges, hypergraphs, hierarchical graphs, ordered graphs, etc.

These graphs make great data structures, but are still not appropriate for information integration. First, it is still impossible to refer to edges (that is, edges are not vertices). The usual attribute workaround adds complexity to the metamodel, especially if composite or vertex attribute values are allowed. Some of the graphs also include a restrictive stratified type system; for example, GXL uses UML. Finally, some of these proposals evolved from the internal data models of graph visualization applications and pollute the metamodel with primitive features specific to graph rendering.

5.4.3. Odds and Ends

Bowers and Delcambre have done some interesting work on superimposed information and metamodel conversion [BD00]. They propose a metamodel based on directed graphs, with a minimal ontology of two special kinds of vertices (literals and references) and two special kinds of edges (classification and generalization). Mappings lift other metamodels into their own, customized to the exact depth of reference needed by the source metamodel. They supply mappings for XML and RDF, and suggest that a mapping for Topic Maps is possible as well. The lifted models are of course not semantically compatible, but this approach concentrates on conversion, not integration.

The Hypernode Model is a nested-graph model employed by Hyperlog [PH01b]. Each hypernode contains a non-well-founded set of hypernodes, and a separate set of directed (but unlabeled) edges. It is quite powerful and has a very well developed formal query and programming language. Programs and the type system are incorporated into the graph. It is probably the metamodel most similar to Braque, but some differences make it inappropriate for information inte-

gration. Hypernodes are strongly typed and have no intrinsic concept of order. Worst of all, the edges cannot be referenced.

There are many other ways to model information. [AP96] reviews the use of sets in information theory. The Notion System (<http://www.notionssystem.com/>) and e4graph (<http://e4graph.sourceforge.net/>) are two graph-like metamodels with working implementations but limited formal documentation. Finally, pure logic representations eschew structural models in favour of direct logical formulas. [SD99] provides a good introduction to these kinds of metamodels; relevant examples include KIF [ANS98] (Knowledge Interchange Format), DAML (the DARPA Agent Markup Language, <http://www.daml.org/>) and Cyc (<http://www.cyc.com/>).

6. Conclusions

This final chapter summarizes and evaluates the contributions made by this thesis, and speculates on fruitful avenues for future work in this area.

6.1. Evaluation and Contributions

The main contribution of this thesis is the Braque metamodel and the integration mappings of XML, RDF and Topic Maps. There are few completely new ideas in Braque: only infinite reflection depth (Section 6.1.1) and the metatype constraint (Section 6.1.4) cannot be found in any other metamodels. The remainder of Braque's innovation lies in recognizing the usefulness of existing features in the context of semantic web information integration, and in constructing a coherent and compact metamodel and upper ontology that implement those features.

Braque fully implements all the features of the IMI object layer—as do most of the other metamodels. However, certain aspects of Braque's design give it superior expressive power: Braque can model constructs that other metamodels either cannot model at all, or only with great difficulty (Sections 6.1.1 and 6.1.2). Section 6.1.3 justifies why Braque is elegant, and explains how it is amenable to a clean implementation in an object-oriented language. Finally, Section 6.1.4 looks at the specific contributions made in the naïve upper ontology and the integration of the three metamodels.

6.1.1. Deep Reflection

Braque's treatment of reification is exceptionally powerful. To understand why, define *reflection depth* as the number of times a non-referenceable model primitive can be successively reified. When a primitive is first reified, it is decomposed into finer primitives without destroying the original. One of those primitives is referenceable and reifies the original. The other primitives encode the information content of the original and are usually non-referenceable. Those primitives can then sometimes be further reified, etc. The maximum number of such de-

composition levels supported by a metamodel is its reflection depth. Furthermore, for the process to qualify as reflection, the primitives that result from reification at each level must be directly linked to the original primitive rather than just be a detached description. (In other words, any changes in the original must be reflected by its reification and vice-versa. Newcomb calls this “preemptive reification” [New02].)

According to this definition, most graph-based metamodels (e.g., OEM, OrdLab Graphs) have a reflection depth of 0, since there is no way to reference edges. RDF also has a reflection depth of 0 since its reified statements are not connected to actual statings. The Yin/Yang metamodel has a reflection depth of 1, since each assertion is represented with a referenceable vertex and two edges. The Topic Maps metamodel has a reflection depth of 2: not only can each association be referenced, but the fact that a subject plays a role in it (a *casting*) also has an identity.

A reflection depth of at least 1 is needed to enable provenance tracking that is not built in as a special mechanism. Newcomb argues [New02] that there is no practical need for reflection more than 2 levels deep. He may well be right; no model in this thesis needs to exceed this limit. However, when integrating metamodels, the origin’s reflection depth cannot exceed that of the target without forcing a lift. This (among other reasons) is why it is impossible to map Topic Maps into RDF without lifting at least one level, to make the mapped associations referenceable.⁴⁵

Braque has an infinite reflection depth (Section 3.3.3), so that metamodels of any depth can be mapped without lift. This maintains an even semantic level across the entire integrated model, which is necessary to ensure that it has a consistent interpretation and can be uniformly queried. As a welcome side effect, the infi-

⁴⁵ A true mapping would need to lift two levels, so that the castings would be referenceable. None of the mappings presented in Section 5.3 do this since Topic Maps have only recently gained their second level of reflection, and no models make references to castings yet.

nite reflection depth also enables easy provenance tracking. It is also more elegant to provide a self-recursive definition for an infinite reflection depth than to define each level explicitly.

To the best of my knowledge, Braque's infinite reflection depth is a unique contribution to metamodel design. Moreover, the explicit definition of reflection depth and the claim that it is a determinant of expressive power appear to be new and could be considered a contribution in their own right.

6.1.2. Expressive Power

Braque's implementation of the other object layer features is not unique, but a few further aspects bear calling some attention to. Braque implements ordering as a primitive. This obviates the need for workarounds such as ordinal properties or successor links, which cannot order dense sets and curtail implementation opportunities (see next section). The primitive ordering makes it easy to integrate ordered metamodels such as XML, and allowing partial orders allows ordered and unordered elements to be merged without losing (or adding) information. Some other metamodels have ordering as a primitive, but very few allow partial orders.

Braque models n -ary ($n > 2$) relationships directly, allowing them to be expressed at the same semantic level as binary ones. This is by no means rare or innovative—after all, the relational model is based on n -tuples—but seems to have fallen by the wayside with the current batch of graph-based metamodels.

Picking the right primitives for a metamodel is important, but just as important is choosing which features should *not* be primitive. In Braque, neither identification nor classification is primitive; rather, they are second-order features introduced in the naïve upper ontology. This allows the agent to bring to bear the full power of the model on their application instead of being restricted to primitive-specific functionality. Reasoning, queries, etc., work the same way as for any other relation. Reasoning about identification by URI gives an agent the chance

to deal with ambiguity, an option that is sadly lacking in the current crop of semantic web metamodels. While this thesis does not propose specific strategies for dealing with ambiguous identifiers, it does not need to: agents can improvise solutions using Braque's existing features.

The critiques in Chapter 5 and the integration of three heretofore-unreconciled metamodels provide further evidence of Braque's flexibility and expressive power.

6.1.3. Elegance

Hyper pomsets are a very elegant primitive when implementing a metamodel in an object-oriented language: they map naturally to the concept of collections, an ineluctable component of any programming environment. Collections are a familiar construct to every developer, and the implementation of any metamodel must use them in its programming interface in some way. Letting collections *be* the model strips a layer of indirection, and should make the system's interface easier to understand. While this was borne out in my own experience of using the implementation, the hypothesis has not been validated with a user study of independent developers.

The ordering design also works nicely with an object-oriented implementation. Since ordering is a primitive, its implementation can be independently optimized for each storage medium, and access provided with a choice of iterators, indexed retrieval and the head/tail technique (great for recursion). This contrasts with unordered metamodels where, for each ordered structure, the model's author must choose between ordinal primitives and linked lists. Code must then be written specifically for the chosen data structure, unacceptably increasing coupling to information best left hidden.

Defining ordering as local to each nest, rather than global to the whole model, also gives us a nice locus of control. In the implementation, each nest has responsibility for maintaining the order of its own elements, and can expose any

appropriate interface for manipulating this order. Once again, this should dovetail nicely with developers' intuitive ideas about collections.

Chapter 5 discussed some other ugly features of other metamodels that are missing from Braque, which could be considered as evidence of elegance by omission. In the end, though, elegance is in the eye of the beholder, and only time (and user studies) will tell if Braque has attained this goal.

6.1.4. Integration

The object level mapping of RDF and Topic Maps into Braque presented in this thesis is, to the best of my knowledge, the only complete lossless integration of the two metamodels. RDF and Topic Maps are integrated not just structurally, but also semantically, so that agents do not need to be aware of the boundaries between the amalgamated models. The mapping includes some of the more esoteric features of both metamodels, including RDF containers, RDF reification and Topic Maps scopes, which are ignored by the other extant mappings. Additionally, a very natural integration of XML at a structural level is supported as well, including full modelling of qualified names.

The integration is not perfect. There is no mapping for RDF schema (domain and range specifications), and no way to explicitly represent RDF's open-world assumption. The ad-hoc solution to class punning is also somewhat ugly and needs to be reviewed.

The process of developing the integration mappings also resulted in some incidental contributions. The analysis of the different kinds of mappings relative to the IMI Reference Model, and the explanation why a lift should not count as integration, apparently have not been put in writing before. The discovery that a metatype constraint can be defined uniformly for all uses of the extension relation and its formulation for an unstratified, multiple classification metamodel are also new.

6.2. Future Work

This thesis has only scratched the surface of semantic web metamodels and metamodel integration. The work presented can be greatly extended, both in depth and in breadth. This section mentions only a few possibilities for future research.

6.2.1. Theory

Hyper pomsets are a powerful metamodel primitive, but they could still be improved. For one, not everything is easily modeled as a nest with discrete members. The theory of mereology deals with constructs that can be subdivided but don't necessarily consist of discrete elements. For example, consider the concept of knowledge: it is possible to possess some knowledge, to accumulate knowledge, etc., but we do not think of knowledge as a set of "knowledge atoms". Mereotopology is a related discipline that applies mereology to the study of space by considering whole-part relationships between shapes rather than thinking of them as made up of an infinite number of dimensionless points. It would be interesting to see whether mereology and mereotopology can be integrated into the same semantic framework as hyper pomsets.

Another issue is that when reasoning on the semantic web an agent is almost never in possession of all the relevant information. In terms of nests, this means that the agent may now know the exact membership of a nest (e.g. a classifier or a relation). If an element is not in the nest, does it mean that the nest definitely does not contain the element (the closed-world assumption), or simply that the agent doesn't know whether the nest contains the element (the open-world assumption)? Normally, a metamodel makes only one of these assumptions to provide a consistent interpretation for its constructs. (For example, RDF makes the open-world assumption.) Instead of making a global assumption for Braque, it would be interesting to look at partial sets, which—in contrast with "crisp" sets—permit an element to *maybe* be part of a set. The closed- vs. open-world de-

cision could then be made on a nest-by-nest basis: if a nest was known to be closed it would be crisp, otherwise it would *maybe* contain every element that was not known to be definitely in or out of the nest. Partial sets may also help find a better solution to the problem of class punning (see Appendix C.4) and may have other interesting applications in fuzzy logic, and in representing the consequences of logical paradoxes.

Braque is also in dire need of more formality. Hyper pomsets (or partial hyper mereo-pomsets, if the suggestions above are adopted) need to be formally defined, perhaps with an appropriate axiomatic theory. The meaning of the meta-model also needs to be formalized, probably by building a model theory for Braque, though an axiomatization that maps it to some other logic (e.g., F-Logic [KLW95], or some description logic) might be another option. Without a semantic interpretation, most logicians would consider Braque to be a mere data model. Mutation of the model should also be formalized with an algebra for the pomsets [GM95], and of course reified within the model itself to complete the current static reflection.

Every self-respecting data model also needs a query language, since structural navigation becomes impractical as model size increases. For Braque, it would be useful to have a more powerful logical inference language that could express the equations used throughout this thesis with a simpler syntax; Equation 4-13, for one, should not look so convoluted. Queries are then just a special case of inference. It would be interesting to explore the many possible approaches to an inference language (e.g., algebra, constraint systems) and compare their expressive power and user friendliness. Of course, both the results of the inferences and the inference language itself ought to be represented within the Braque metamodel—this is likely to increase the languages' power, but may also make the implementation intractable. Lastly, the user must be given control over the selective application of inferences: not all equations hold globally.

The naïve upper ontology will also need to be updated to keep pace with these improvements. With an inference engine available, work can begin on integrating the various schema constraint facilities offered by other metamodels. Inference could also be used for synthetic class constructors (favoured by description logic mavens) and property inheritance (beloved of object-oriented programmers). Finally, the metatype constraint (Equation 3-12) and the difference between expansion and extension should be investigated further. Other metamodels must be surveyed to determine whether similar constraints are present and what forms they take, as well as the consequences of these design decisions. Further study could validate the generalized form of the constraint presented here and point out hidden flaws in other metamodels.

6.2.2. Integration

The integration of XML, RDF and Topic Maps presented in this thesis is just a beginning. Integrating natural language scoping is one obvious improvement that could be done right away. Providing a (partial) mapping from Braque back into the other metamodels would enable Braque to be used for model translation. Once Braque has a model theory, it would also be important to show that the mappings preserve meaning by proving that, for every model, the original model theory and Braque's model theory give the same interpretation.

There is also more work to be done on the integration mappings themselves. Since RDF and Topic Maps are both still evolving, their Braque mappings need to track this development. The structural XML mapping should also be augmented with some semantics, perhaps derived from an external description in a language such as XML Schema or the Meaning Definition Language (MDL) [Wor01]. Ontologies and mappings could also be devised for constructs defined by XPath, XPointer, and XLink.

Naturally, there is a nearly endless supply of other metamodels that could be integrated. While the specific choices will be driven by application needs, a few particularly interesting selections include:

- UML [OMG01], and its meta-metamodel the Meta-Object Facility (MOF) [OMG02], are a pair of stratified metamodels popular in object-oriented software development and other modeling endeavours. Many other metamodels conform to the MOF, so integrating these metamodels would give Braque access to a great deal of instance data.
- The web services effort is spurning RDF and developing the Universal Description, Discovery and Integration language (UDDI) [BC+02] instead. Clients can query service traders to locate appropriate services whose functions are described with a UDDI model. This is a burgeoning area (both of research and practical implementation) and giving Braque and understanding of UDDI models might open other opportunities in the future.

6.2.3. Implementation

Just like every other proof-of-concept, the implementation written for this thesis could use a good redesign and a lot of optimization, but this section focuses on future work specific to Braque.

The current implementation was written in Java, a conventional object-oriented programming language. Ideas are instances of an *Idea* class, classifiers are instances of a *Nest* class whose members are ideas, etc. The semantics of the Braque metamodel and Java are completely disjoint; in effect, the Braque world is using Java's object system as syntax. This suggests an intriguing possibility: can Braque's semantics be aligned with those of a sufficiently flexible object-oriented language, so that the language itself becomes the object layer and Braque fills in the gaps with a thin semantic layer on top? Such an amalgamation would require the language either to be classless, or to sport an unstratified type system with multiple dynamic classification, at a minimum. Object-oriented lan-

guages that fit these requirements are rare, and the integration is likely to be technically tricky, but the effort should be worth it: instead of interacting with the model through APIs, Braque would become the language, making the programming interface seamless.

Leaving this radical approach aside for now, a conventional implementation could also be improved in many ways:

- Once two-way mappings are available, the knowledge base could emulate common APIs of the other metamodels, thus enlarging the pool of software able to manipulate a Braque model.
- Designing and implementing a modular, complete and efficient inference engine is a multi-year research project in and of itself.
- Right now, the Braque knowledge base needs to import and export documents to share information. To truly establish a semantic web, the knowledge base should be endowed with a direct communication interface. Some options include an HTTP-based resource gateway where each idea is given a URI and interaction proceeds through HTTP methods, a SOAP server that exposes a custom API, and a peer-to-peer distributed knowledge network.

Finally, both the current implementation and any future ones need extensive usability studies to determine what programming interface developers prefer.

6.2.4. Follow-On Work

While the previous sections dealt with further work on the Braque metamodel itself and its intended use as an integration platform, many other applications could be built on such a flexible metamodel core. A generic model visualization and manipulation utility would be a good start, enabling end users to directly control the structure of a model. A more advanced visualization framework design would allow additional renderers to be plugged in dynamically, embedding specialized displays adapted to the extended semantics of parts of the model.

Though there is a lot of previous work in this area, most concentrates on visualizing simple directed graphs or object-like frames that follow strict schemas.

Other domain-specific applications of Braque include:

- modelling multi-dimensional programs—the original motivation for Braque’s inception (as described in Appendix C.1);
- reasoning about concurrent processes, which can be modelled with pomsets;
- storing the knowledge an automated agent has acquired while going about its tasks on the semantic web; and
- helping people deal with information overload by letting them easily record, organize, link and annotate all the information they acquire.

References

All references to W3C documents list the URI of the “latest version”, since the URI is shorter and the most recent revision of a recommendation is likely to be of more interest than the exact one in effect at the time of writing of this thesis. However, should you wish to see this older version, you can follow the link from the current document to the older one identified by the publication date given in the reference.

- [Acz88] Peter Aczel: *Non-Well-Founded Sets*. CSLI Publications, Stanford, 1988.
- [ANS98] *Knowledge Interchange Format*. Draft proposed North American Standard, NCITS.T2/98-004. <http://logic.stanford.edu/kif/dpans.html>
- [AO+02] Pascal Auillans, Patrice Ossona de Mendez, Pierre Rosenstiehl and Bernard Vatant: *A Formal Model for Topic Maps*. Proceedings of the First International Semantic Web Conference, Sardinia, Italy, June 2002.
- [AP96] Varol Akman and Müjdat Pakkan: *Nonstandard Set Theories and Information Management*. Journal of Intelligent Information Systems, Volume 6 Number 1, 1996, pp. 5-31.
- [ASB99] Serge Abiteboul, Dan Suciu, and Peter Buneman: *Data on the Web: From Relations to Semistructured Data and XML*. Morgan Kaufmann Publishers, October 1999.
- [BC+02a] Scott Boag, Don Chamberlin, Mary Fernández, Daniela Florescu, Jonathan Robie and Jérôme Siméon, editors: *XQuery 1.0: An XML Query Language*. W3C Working Draft, August 2002. <http://www.w3.org/TR/xquery/>
- [BC+02b] Tom Bellwood, Luc Clément, David Ehnebuske et al., editors: *UDDI Version 3.0*. Published Specification, July 2002. http://uddi.org/pubs/uddi_v3.htm
- [BD00] Shawn Bauers and Lois Delcambre: *Representing and Transforming Model-Based Information*. Proceedings of the Semantic Web Workshop 2000, Lisbon, Portugal, September 2000. <http://www.ics.forth.gr/proj/isst/SemWeb/proceedings/session1-1/paper.pdf>
- [BE+01] Ulrik Brandes, Markus Eiglsperger, Ivan Herman, Michael Himsolt, and M. Scott Marshall: *GraphML Progress Report: Structural Layer Proposal*. Proceedings of Ninth International Symposium on Graph Drawing, Vienna, September 2001. <http://www.inf.uni-konstanz.de/~brandes/publications/behhm-gprsl-01.pdf>

- [Bec02] Dave Beckett, editor: *RDF/XML Syntax Specification (Revised)*. W3C Working Draft, March 2002. <http://www.w3.org/TR/rdf-syntax-grammar/>
- [Ber98] Tim Berners-Lee: *Why RDF model is different from the XML model*. Design Issues draft, October 1998.
- [BG02] Dan Brickley and R. V. Guha, editors: *RDF Vocabulary Description Language 1.0: RDF Schema*. W3C Working Draft, April 2002. <http://www.w3.org/TR/rdf-schema/>
- [BHL01] Tim Berners-Lee, James Hendler, Ora Lasilla. *The Semantic Web*. Scientific American, May 2001.
- [BHL99] Tim Bray, Dave Hollander and Andrew Layman, editors: *Namespaces in XML*. W3C Recommendation, January 1999. <http://www.w3.org/TR/REC-xml-names/>
- [BM+98a] Tim Berners-Lee, MIT/LCS, Roy Fielding, U.C. Irvine, Larry Masinter and Xerox Corporation: *Uniform Resource Identifiers (URI): Generic Syntax*. The Internet Society, RFC 2396, 1998.
- [BM+98b] William J. Brown, Raphael C. Malveau, Hays W. McCormick III and Thomas J. Mowbray: *Anti Patterns: Refactoring Software, Architectures, and Projects in Crisis*. John Wiley & Sons, New York, 1998.
- [BM01] Paul V. Biron and Ashok Malhotra, editors: *XML Schema Part 2: Datatypes*. W3C Recommendation, May 2001. <http://www.w3.org/TR/xmlschema-2/>
- [Bol01] Harold Boley: *A Web Data Model Unifying XML and RDF*. Draft, September 2001. <http://www.dfki.uni-kl.de/~boley/xmlrdf.html>
- [Bou00] Ronald Bourret: *Namespace Myths Exploded*. XML.com, O'Reilly, March 2000. <http://www.xml.com/pub/a/2000/03/08/namespaces/index.html>
- [BPS00] Tim Bray, Jean Paoli, C. M. Sperberg-McQueen, Eve Maler, editors: *Extensible Markup Language (XML) 1.0, 2nd ed.* W3C Recommendation, October 2000. <http://www.w3.org/TR/REC-xml>
- [Bri02] Dan Brickley: *RDF: Understanding the Striped RDF/XML Syntax*. Version 1.30, August 2002. <http://www.w3.org/2001/10/stripes/>
- [Car60] Lewis Carroll: *The Annotated Alice With an Introduction and Notes by Martin Gardner*. Bramhall House, 1960, Through the Looking-Glass, Chapter 8, pp. 306-307.
- [CD99] James Clark and Steven DeRose, editors: *XML Path Language (XPath) Version 1.0*. W3C Recommendation, November 1999. <http://www.w3.org/TR/xpath>
- [Cla99] James Clark, editor: *XSL Transformations (XSLT) Version 1.0*. W3C Recommendation, November 1999. <http://www.w3.org/TR/xslt>

- [CM01] James Clark and Murata Makoto, editors: *RELAX NG Specification*. OASIS Committee Specification, December 2001. <http://www.oasis-open.org/committees/relax-ng/spec-20011203.html>
- [Cov98] Robin Cover: *XML and Semantic Transparency*. Cover Pages, November 1998. <http://xml.coverpages.org/xmlAndSemantics.html>
- [CT01] John Cowan and Richard Tobin: *XML Information Set*. W3C Recommendation, October 2001. <http://www.w3.org/TR/xml-infoset/>
- [DC+02] Mike Dean, Dan Connolly, Frank van Hermelen et al., editors: *OWL Web Ontology Language 1.0 Reference Description*. Draft document, June 2002. <http://lists.w3.org/Archives/Public/www-webont-wg/2002Jun/att-0124/01-owl-ref-proposed.html>
- [DMD01] Steven DeRose, Eve Maler and Ron Daniel Jr., editors: *XML Pointer Language (XPointer) Version 1.0*. W3C Recommendation, September 2001. <http://www.w3.org/TR/xptr/>
- [DMO01] Steve DeRose, Eve Maler and David Orchard, editors: *XML Linking Language (XLink) Version 1.0*. W3C Recommendation, June 2001, <http://www.w3.org/TR/xlink/>.
- [Doc01] Corey Doctorow: *Metacrap: Putting the torch to seven straw-men of the meta-utopia*. August 2001. <http://www.well.com/~doctorow/metacrap.htm>
- [Dum02] Edd Dumbill: *Finding friends with XML and RDF*. IBM developerWorks, XML Watch column, June 2002. <http://www-106.ibm.com/developerworks/xml/library/x-foaf.html>
- [Ebe02] Andreas Eberhart: *Survey of RDF Data on the Web*. Technical Report, International University in Germany, August 2002. <http://www.i-u.de/schools/eberhart/rdf/rdf-survey.pdf>
- [EF94] Jürgen Ebert and Angelika Franzke: *A Declarative Approach to Graph Based Modeling*. Technical Report 3-94, Universität Koblenz-Landau, 1994.
- [Euz02] Jérôme Euzanat, editor: *Research challenges and perspectives of the Semantic Web*. Report of the EU-NSF strategic workshop, Sophia-Antipolis, France, January 2002. <http://www.ercim.org/EU-NSF/semweb.html>
- [Fie99] Roy Fielding et al.: *Hypertext Transfer Protocol – HTTP/1.1*. Internet Engineering Task Force, Request For Comments 2616, Draft Standard, June 1999.
- [FMG02] Mary Fernández, Jonathan Marsh and Marton Nagy, editors: *XQuery 1.0 and XPath 2.0 Data Model*. W3C Working Draft, August 2002. <http://www.w3.org/TR/query-datamodel/>
- [For02] Paul Ford: *August 2009: How Google beat Amazon and Ebay to the Semantic Web*. July 2002. http://ftrain.com/google_takes_all.html

- [Fow00] Martin Fowler with Kendall Scott: *UML Distilled Second Edition*. Addison-Wesley, 2000.
- [Gar01a] Lars Marius Garshol: *tolog: A topic map query language*. Proceedings of XML Europe 2001, Berlin, May 2001. <http://www.ontopia.net/topicmaps/materials/tolog.html>
- [Gar01b] Lars Marius Garshol: *Topic maps, RDF, DAML, OIL: A comparison*. Proceedings of XML Conference & Exposition 2001, Orlando, Florida, December 2001. <http://www.idealliance.org/papers/xml2001/papers/html/05-04-04.html>
- [Gar02] Lars Marius Garshol: *The Linear Topic Map Notation: Definition and introduction, version 1.2*. Ontopia A/S, May 2002. <http://www.ontopia.net/download/ltn.html>
- [GB02] Jan Grant and Dave Beckett, editors: *RDF Test Cases*. W3C Working Draft, April 2002. <http://www.w3.org/TR/rdf-testcases/>
- [GM02] Lars Marius Garshol and Graham Moore: *The Standard Application Model for Topic Maps*. ISO JTC1/SC34 N329, July 2002. <http://www.y12.doe.gov/sgml/sc34/document/0329.htm>
- [GM95] Stephane Grumbach and Tova Milo: *An Algebra for Pomsets*. Proceedings of the Fifth International Conference on Data Base Theory (ICDT'95), Prague, Lecture Notes in Computer Science 893, pages 191-207, Springer-Verlag, 1995.
- [GP+94] Marc Gyssens, Jan Paredaens, Jan Van den Bussche, and Dirk van Gucht: *A Graph-Oriented Object Database Model*. IEEE Transactions on Knowledge and Data Engineering, Vol. 6, No. 4, 1994, pp. 572-586.
- [GR89] Adele Goldberg and David Robson: *Smalltalk-80 The Language*. Addison-Wesley, 1989, p. 247.
- [Gra02] Marc de Graauw: *Note on issues to be decided on scope*. ISO JTC1/SC34 N327, July 2002. <http://www.y12.doe.gov/sgml/sc34/document/0327.htm>
- [Gri82] Robert L. Griffith: *Three Principles of Representation for Semantic Networks*. ACM Transactions on Database Systems, Vol. 7, No. 3, September 1982, pp. 417-442.
- [Haw02] Sandro Hawke: *Explicit disambiguation via RDF bNodes, more process*. RDF-interest mailing list, April 2002. <http://lists.w3.org/Archives/Public/www-rdf-interest/2002Apr/0327.html>
- [Hay02a] Patrick Hayes, editor: *RDF Model Theory*. W3C Working Draft, April 2002. <http://www.w3.org/TR/rdf-mt/>
- [Hay02b] Patrick Hayes: *rdfs:subClassOf and metaclasses*. RDF-comments mailing list, July 2002. <http://lists.w3.org/Archives/Public/www-rdf-comments/2002JulSep/0017.html>

- [HMS02] Patrick Hayes, Sergey Melnik and Patrick Stickler, editors: *RDF Datatyping*. W3C Working Draft, April 2002. <http://www-nrc.nokia.com/sw/rdf-datatyping.html>
- [How01] Denis Howe, ed. *The Free On-line Dictionary of Computing*. <http://www.foldoc.org/>
- [ISO86] ISO JTC1 SC34: *Standard Generalized Markup Language*. ISO 8879:1986, 1986.
- [ISO99] ISO JTC1 SC34: *Topic Maps*. ISO 13250:2000, December 1999.
- [Kal02] Aditya Kalyanpur: *RDF Web Scraper*. <http://www.ece.umd.edu/~adityak/running.html>, visited June 29, 2002.
- [KC02] Graham Klyne and Jeremy Carroll, editors: *Resource Description Framework (RDF): Concepts and Abstract Data Model*. W3C Working Draft, August 2002. <http://www.w3.org/TR/rdf-concepts/>
- [Kle01] Kevin C. Klement: *Russell's Paradox*. In *The Internet Encyclopedia of Philosophy*, edited by James Fieser and Bradley Dowden, 2001. <http://www.utm.edu/research/iep/p/par-russ.htm>
- [KLW95] Michael Kifer, Georg Lausen and James Wu: *Logical Foundations of Object-Oriented and Frame-Based Languages*. *Journal of the ACM*, volume 42, pp. 741-783, 1995.
- [LD01] Martin S. Lacher, Stefan Decker: *On the Integration of Topic Maps and RDF Data*. Proceedings of the Semantic Web Working Symposium 2001, Stanford University, California, USA, July 2001.
- [Lis88] Barbara Liskov: *Data Abstraction and Hierarchy*. *SIGPLAN Notices*, 23(5), May 1988.
- [LS99] Ora Lassila and Ralph R. Swick, editors: *Resource Description Framework (RDF) Model and Syntax Specification*. W3C Recommendation, February 1999. <http://www.w3.org/TR/REC-rdf-syntax>
- [MA+02] Aimilia Magkanaraki, Sofia Alexaki, Vassilis Christophides and Dimitris Plexousakis: *Benchmarking RDF Schemas for the Semantic Web*. Proceedings of the First International Semantic Web Conference, Sardinia, Italy, June 2002.
- [MA+02] Shane McCarron, Jonny Axelsson, Beth Epperson, Ann Navarro and Steven Pemberton, editors: *XHTML 2.0*. W3C Working Draft, August 2002. <http://www.w3.org/TR/xhtml2/>
- [Mas02] Larry Masinter: *"duri" and "tdb" URN namespaces based on dated URIs*. Internet Draft, work in progress, April 2002. <http://larry.masinter.net/duri.html>

- [MD00] Sergey Melnik and Stefan Decker: *A Layered Approach to Information Modeling and Interoperability on the Web*. Proceedings of the ECDL'00 Workshop on the Semantic Web, Lisbon, Portugal, September 2000.
- [Mel99] Sergey Melnik: *Bridging the Gap between RDF and XML*. December 1999. <http://www-db.stanford.edu/~melnik/rdf/fusion.html>
- [Men83] Valerie Mendes, project editor: *A History of Art*. Macmillan London, 1983, p. 850.
- [Mil01] George A. Miller, principal investigator: *WordNet 1.7.1*. Princeton University, 2001. <http://www.cogsci.princeton.edu/~wn/>
- [Moo01] Graham Moore: *RDF and TopicMaps: An Exercise in Convergence*. Proceedings of XML Europe 2001, Berlin, May 2001.
- [Moo02] Graham Moore: *An Integration of TopicMaps and RDF Using N3, Dublin Core and RDFS*. Final Draft, unpublished & undated, assumed to have been written in 2002. <http://www.semanticwebserver.com/papers/integration2.doc>
- [Nag56] Ernest Nagel: *Symbolic Notation, Haddocks' Eyes and the Dog-Walking Ordinance*. The World of Mathematics volume three, edited by James R. Newman, Simon and Schuster, New York, 1956, pp. 1878-1900.
- [NB01] Steven R. Newcomb and Michael Biezunski: *Topicmaps.net's Processing Model for XTM 1.0, version 1.02*. July 2001. <http://www.topicmaps.net/pmtrm4.htm>
- [NB02] Steven R. Newcomb and Michael Biezunski, editors: *A Draft Reference Model for ISO 13520 Topic Maps*. ISO JTC1/SC34 N298 Rev. 1, April 2002. <http://www.y12.doe.gov/sgml/sc34/document/0298R1.htm>
- [New02] Steven R. Newcomb: *Preemptive Reification*. Proceedings of International Semantic Web Conference, Sardinia, Italy, June 2002.
- [NWC00] Wolfgang Nejdl, Martin Wolpers and Christian Capelle: *The RDF Schema Specification Revisited*. Proceedings of Modellierung 2000, St. Goar, Germany, April 2000.
- [Ogi01] Nikita Ogievitsky: *XML Topic Maps through RDF Glasses*. Presented at Extreme Markup Languages 2001, Montréal, Québec, August 2001. <http://www.cogx.com/xtm2rdf/extreme2001/>
- [OMG01] Object Management Group: *OMG Unified Modeling Language Specification*. Version 1.4, September 2001. <http://www.omg.org/technology/documents/formal/uml.htm>.
- [OMG02] Object Management Group: *Meta Object Facility (MOF) Specification*. Version 1.4, April 2002. <http://www.omg.org/technology/documents/formal/mof.htm>

- [Pep00] Stever Pepper: *Topic maps and RDF: A first cut*. Ontopia AS, June 2000. <http://www.ontopia.net/topicmaps/materials/rdf.html>
- [Pep01] Steve Pepper: *Draft requirements, examples, and a "low bar" proposal for Topic Map Constraint Language*. ISO JTC1/SC34 N226, May 2001. <http://www.y12.doe.gov/sgml/sc34/document/0226.htm>
- [PG01] Stever Pepper and Geir Ove Grønmo: *Towards a General Theory of Scope*. Proceedings of Extreme Markup Languages 2001, Montréal, Canada, August 2001. <http://www.ontopia.net/topicmaps/materials/scope.htm>
- [PGW95] Yannis Pakakonstantinou, Hector Garcia-Molina, Jennifer Widom: *Object Exchange Across Heterogeneous Information Sources*. Proceedings of the International Conference on Data Engineering, Taipei, Taiwan, 1995.
- [PH01a] Jeff Z. Pan and Ian Horrocks: *Metamodeling architecture of web ontology languages*. Proceedings of the Semantic Web Working Symposium, Stanford, July 2001, pp. 131-149.
- [PH01b] Alexandra Poulouvasilis and Stefan G. Hild: *Hyperlog: A Graph-Based System for Database Browsing, Querying, and Update*. IEEE Transactions on Knowledge and Data Engineering, Vol. 13, No. 2, March/April 2001, pp. 316-332.
- [PK01] John Punin and Mukkai Krishnamoorthy, editors: *XGMML 1.0 Draft Specification*. June 2001. <http://www.cs.rpi.edu/~puninj/XGMML/draft-xgmml.html>
- [PM01] Steve Pepper and Graham Moore, editors: *XML Topic Maps (XTM) 1.0*. TopicMaps.org specification, August 2001. <http://www.topicmaps.org/xtm/1.0/>
- [PS02a] Peter Patel-Schneider and Jérôme Siméon: *The Yin/Yang Web: XML Syntax and RDF Semantics*. Proceedings of the 11th International World Wide Web Conference, May 2002. <http://www2002.org/CDROM/refereed/231/>
- [PS02b] Peter Patel-Schneider and Jérôme Siméon: *Building the Semantic Web on XML*. Proceedings of the First International Semantic Web Conference, Sardinia, Italy, June 2002.
- [RHJ99] Dave Raggett, Arnaud Le Hors and Ian Jacobs, editors: *HTML 4.01 Specification*. W3C Recommendation, December 1999. <http://www.w3.org/TR/html4/>
- [RM02] Martin Robillard and Gail Murphy: *Concern Graphs: Finding and Describing Concerns Using Structural Program Dependencies*. Proceedings of International Conference on Software Engineering (ICSE) 2002, Orlando, Florida, May 2002. <http://www.cs.ubc.ca/labs/spl/papers/2002/icse02-feat.pdf>

- [Rob01] Jonathan Robie: *The Syntactic Web: Syntax and Semantics on the Web*. Proceedings of XML Conference & Exposition 2001, Orlando, Florida, December 2001. <http://www.idealliance.org/papers/xml2001/papers/html/03-01-04.html>
- [Sch02] Guus Schreiber: *A UML Presentation Syntax for OWL Lite*. Internal Webont draft, April 2002. <http://www.swi.psy.uva.nl/usr/Schreiber/docs/owl-uml/owl-uml.html>
- [SD99] John F. Sowa and David Dietz: *Knowledge Representation: Logical, Philosophical, and Computational Foundations*. Brooks/Cole, 1999.
- [ST99] Ralph R. Swick and Henry S. Thompson, editors: *The Cambridge Communiqué*. W3C Note, October 1999. <http://www.w3.org/TR/1999/NOTE-schema-arch-19991007>
- [Sti01] Patrick Stickler: *X-Values: Typed Data Literals for the Semantic Web and Beyond*. Nokia Research Center, Software Technology Laboratory, Agent Technology Group, October 2001. http://www-nrc.nokia.com/sw/X_Values_URI.pdf
- [Sto98] Margaret-Anne D. Storey: *A Cognitive Framework for Describing and Evaluating Software Exploration Tools*. PhD Thesis, School of Computing Science, Simon Fraser University, December 1998.
- [TB+01] Henry S. Thompson, David Beech, Murray Maloney and Noah Mendelsohn, editors: *XML Schema Part 1: Structures*. W3C Recommendation, May 2001. <http://www.w3.org/TR/xmlschema-1/>
- [TO+99] Peri Tarr, Harold Ossher, William Harrison and Stanley M. Sutton, Jr.: *N Degrees of Separation: Multi-Dimensional Separation of Concerns*. Proceedings of International Conference on Software Engineering (ICSE) 2002, Los Angeles, California, May 1999. <http://www.research.ibm.com/hyperspace/Papers/icse99.ps>
- [Tur96] Jane Turner, editor: *The Dictionary of Art*. Macmillan Publishers, 1996, Volume 4, p. 673.
- [US87] David Ungar and Randall B. Smith: *Self: The power of simplicity*. OOPSLA '87 Conference Proceedings, Orlando, FL, October, 1987, pp. 227-241.
- [Vli00] Eric van der Vlist: *XML Linking Technologies*. O'Reilly XML.com, October 2000. <http://www.xml.com/pub/a/2000/10/04/linking/index.html>
- [Vli01] Eric van der Vlist: *Naïve approach to representing XTM 1.0 as RDF*. RDF Interest Mailing List, March 8, 2001. <http://lists.w3.org/Archives/Public/www-rdf-interest/2001Mar/0054.html>
- [Wei02] Eric Weisstein: *World of Mathematics*. Wolfram Research, 2002. <http://mathworld.wolfram.com/>

- [Wil00] Simon Williams: *The associative model of data*. Lazy Software Ltd., Great Britain, 2000.
- [Win01] Andreas Winter: *Exchanging Graphs with GXL*. Proceedings of Ninth International Symposium on Graph Drawing, Vienna, September 2001. <http://www.gupro.de/winter/Papers/winter2001a.pdf>
- [WK+99] S. Weibel, J. Kunze, C. Lagoze and M. Wolf, editors: *Dublin Core Metadata Element Set, Version 1.1: Reference Description*. Dublin Core Metadata Initiative, July 1999. <http://dublincore.org/documents/dces/>
- [Won98] Kenny Wong: *Rigi User's Manual, Version 5.4.4*. University of Victoria, June 1998. <http://ftp.rigi.csc.uvic.ca/pub/rigi/doc/rigi-5.4.4-manual.pdf>
- [Wor01] Robert Worden: *A Meaning Definition Language*. Draft 2.02, Charteris, May 2001. <http://www.charteris.com/mdl/MDLWhitePaper.pdf>

A. Braque Metamodel Reference

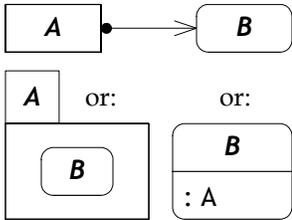
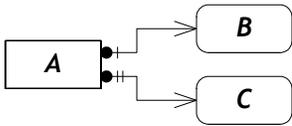
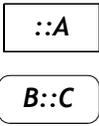
Notation	Explanation
	An anonymous nest with unique identity.
	A nest labeled <i>A</i> . The label is used only for referencing the nest in text and is not part of the model.
	One anonymous and one labeled atom.
	<p><i>A</i> is a nest that contains the atom <i>B</i>. The alternative notations are equivalent. Nests can contain atoms or nests. Atoms cannot contain anything.</p>
	<p><i>A</i> is a nest that contains <i>B</i> and <i>C</i>, in that order. The order of the members is imposed by the increasing number of tic marks.</p>
	<p>References to nest <i>A</i> and atom <i>C</i> from nest <i>B</i>. These do not introduce new unique ideas but rather reference an idea from another diagram. The scopes can be nested to any level.</p>
	<p>Various literal ideas, that uniquely reify the corresponding real-world value. Pictured are strings, numbers and URIs, but more can be added. The type of each literal is implied by the syntax. The type is a nest contained in <i>Domains</i>.</p>

Figure A-1. Braque metamodel graphical notation primitives

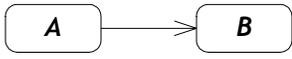
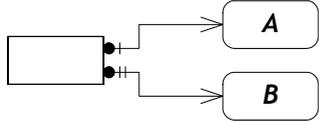
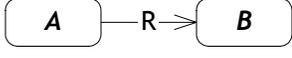
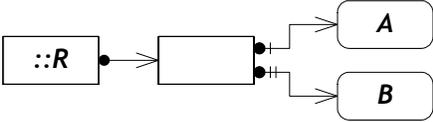
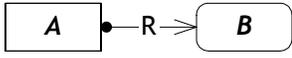
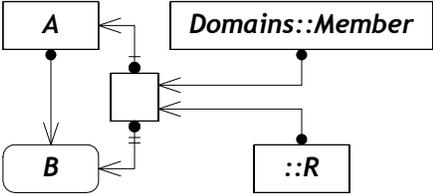
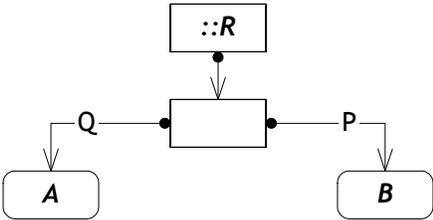
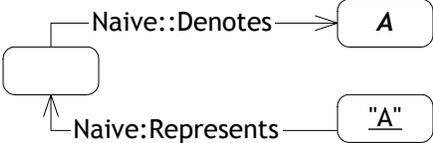
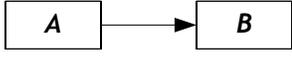
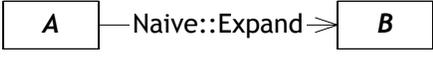
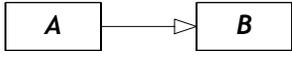
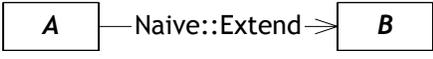
Notation	Explanation	Expansion
	<p>An anonymous, unclassified binary relator from <i>A</i> to <i>B</i>.</p>	
	<p>An anonymous binary relator from <i>A</i> to <i>B</i>, that is also a member of the nest <i>R</i> (usually a binary relation).</p>	
	<p><i>A</i> contains <i>B</i>, and the reified membership pair is a member of <i>R</i> (usually a role).</p>	
	<p>An anonymous binary relationship between <i>A</i> and <i>B</i>, that is also a member of the nest <i>R</i> (usually a relation). <i>P</i> and <i>Q</i>, if present, give the roles played by the membership reification pairs of <i>A</i> and <i>B</i>, respectively.</p>	
	<p>An atom called "<i>A</i>". The idea's name is also used as a label when referring to it. The same notation applies to nests.</p>	
	<p><i>A</i> expands <i>B</i>.</p>	
	<p><i>A</i> extends <i>B</i>. Both <i>A</i> and <i>B</i> must be <i>Classifiers</i>.</p>	

Figure A-2. Braque metamodel graphical notation shorthands

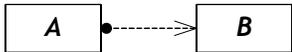
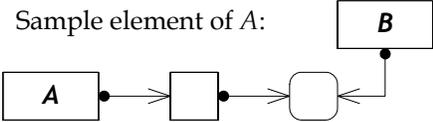
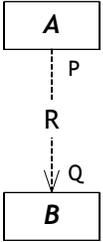
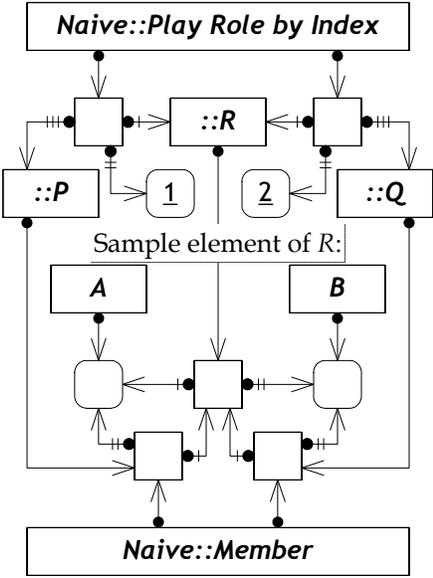
Notation	Explanation	Expansion
	<p>Members of <i>A</i> are nests that may contain members of <i>B</i>. Other kinds of members may also be allowed.</p>	<p>Sample element of <i>A</i>:</p> 
	<p>The binary relation <i>R</i> relates elements of <i>A</i> to elements of <i>B</i>. Other elements may also be allowed. The members of instances of <i>R</i> at index 1 play role <i>P</i>, and those at index 2 play role <i>Q</i>. If <i>P</i> or <i>Q</i> is missing, ignore the part of the expansion that uses it.</p>	

Figure A-3. Braque metamodel graphical notation relation hints

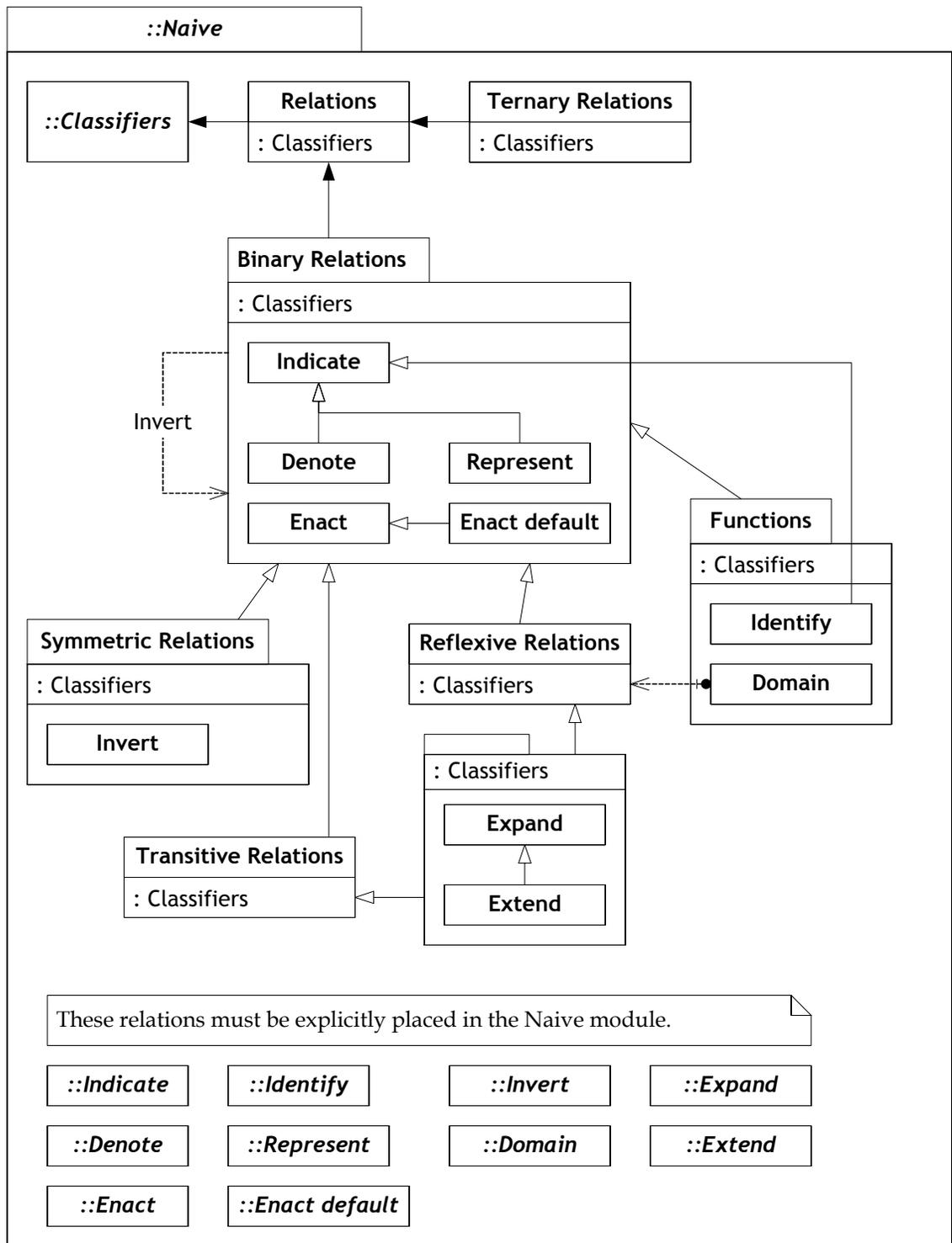


Figure A-5. Naive upper ontology relations

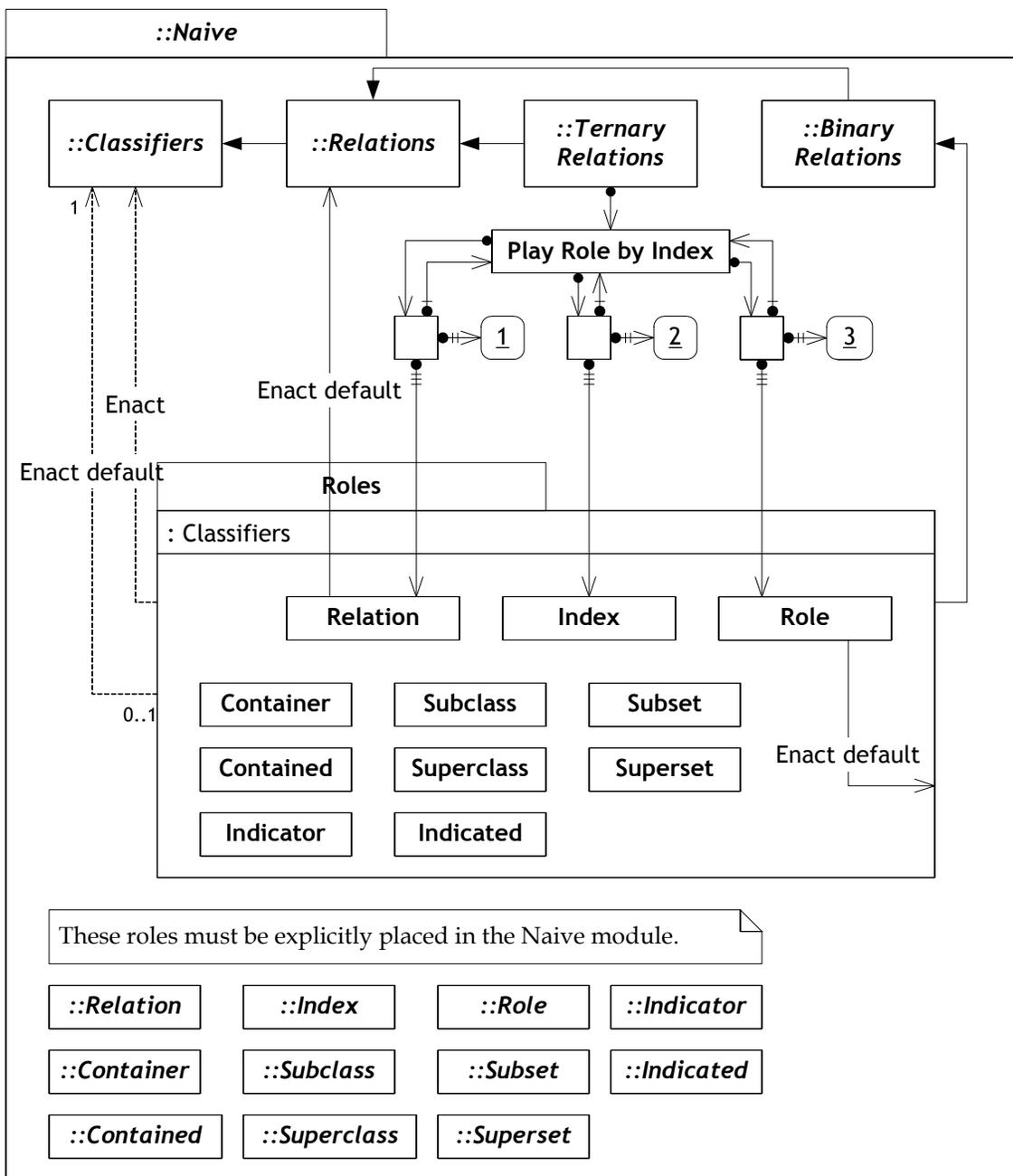


Figure A-6. Naive upper ontology roles

B. Integration Metamodels Reference

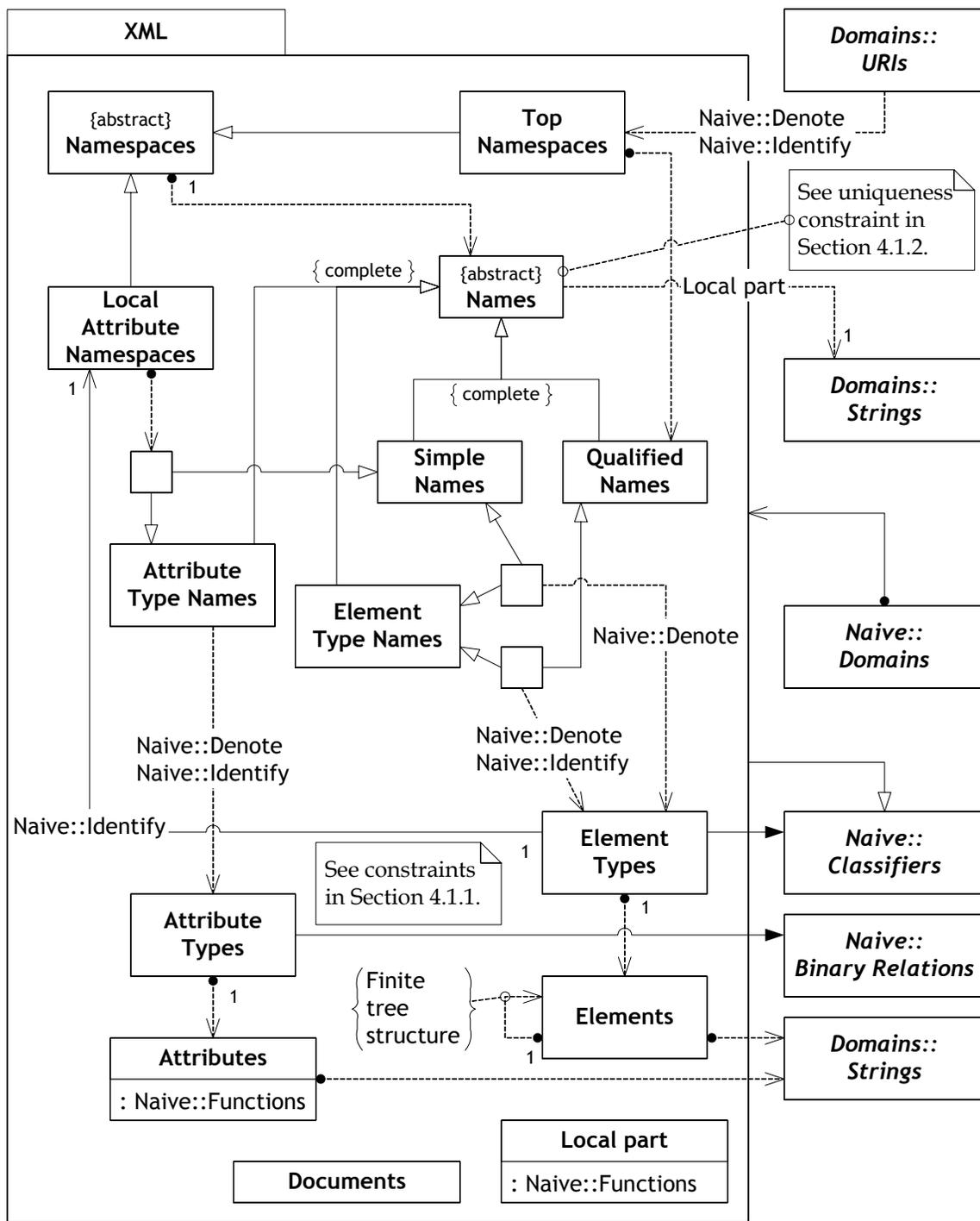


Figure B-1. XML mapping ontology

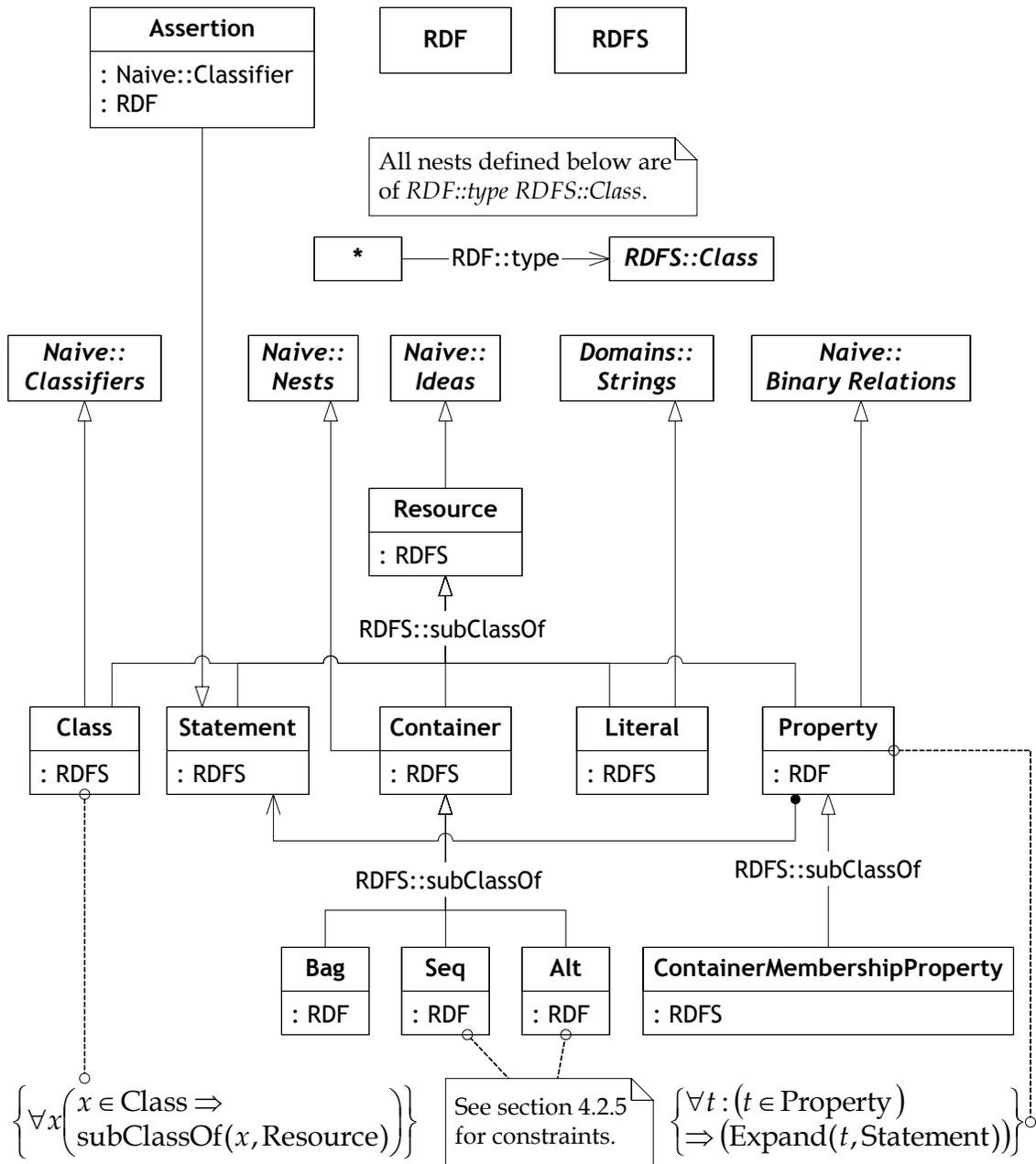


Figure B-2. RDF and RDFS classes

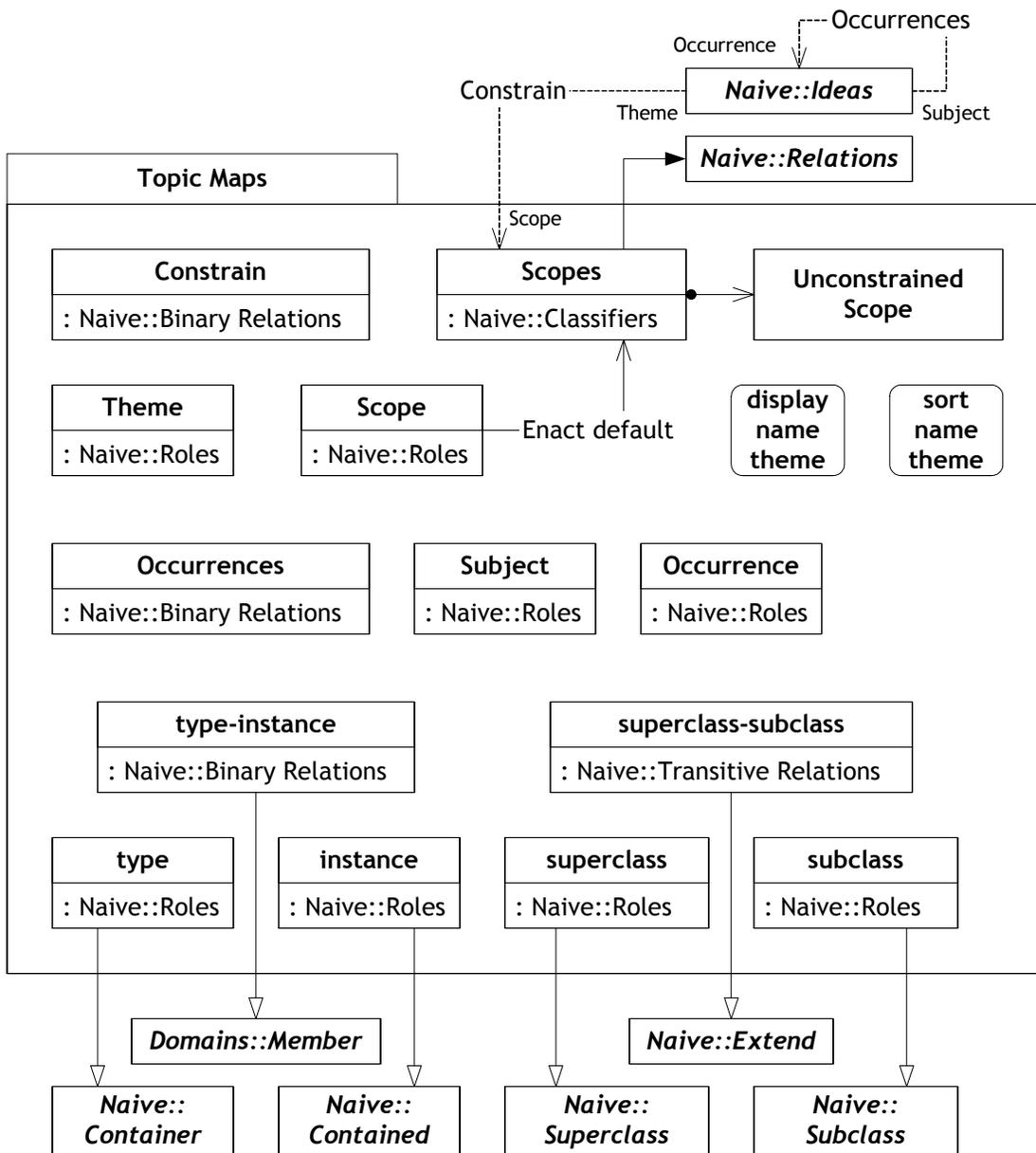


Figure B-4. Topic Maps mapping ontology

C. Research Notes

C.1. Braque Background

The Braque metamodel is named after Georges Braque (1882-1963), a French artist. Georges was a contemporary of the more famous Pablo Picasso. Around 1910, they collaborated to found the Cubist movement. Most art historians give them equal credit for this innovation, but Georges eschewed the limelight and so only Picasso's name is remembered by laypeople [Tur96].

In Cubism, Pablo and Georges rejected the artifice of naturalistic illusionism by simultaneously depicting an object from multiple viewpoints and at multiple times. "The Cubists intended painting to take account of the shifting, sometimes irrational and random, nature of human experience of things and places and of time and space." [Men83] Cubist paintings are easy to recognize (see Figure C-1 for an example), though their disconnected nature does not endear them to the casual viewer.

I was unable to obtain an acceptable copyright license for this picture for purposes of web publication. Please obtain a printed copy of this thesis from the University of Victoria library, or try searching the web for [Braque's Still Life with Violin and Pitcher](#).

Figure C-1. G. Braque's "Still Life with Violin and Pitcher" (1909-10)
© Estate of Georges Braque / SODRAC (Montréal) 2002

How is the Braque metamodel related to cubism? The project's original purpose was to extend reverse engineering and code visualization systems to support new programming languages that employ multidimensional separation of concerns (MDSOC) [TO+99][RM02]. MDSOC advocates decomposing a system along multiple dimensions simultaneously, isolating orthogonal concerns to prevent tangling and scattering and thus enhance maintainability. The pieces are later reintegrated into a coherent system that combines the various points of view. The correspondence to Cubism is obvious, at least at this level of abstraction. However, to name the project directly after Cubism would have been too blunt, and Picasso's name is too well known. Braque, with his short, obscure name, was a better choice.

The project has since drifted from its roots. I found the current tools' metamodels to be inadequate for the multidimensional models I needed to build, and set

out to create a more powerful metamodel. The application to the emerging semantic web was opportunistic, but a good fit in retrospect. The project's name has remained throughout these perambulations, but, coming full circle, the "shifting, sometimes irrational and random, nature of human experience" might now be an apt description of the loosely woven semantic web that Braque tries to integrate.

C.2. Subclass or Instance?

When should a concept be a subclass and when should it be an instance? This question is often confusing to object-orientation neophytes and much time is spent explaining the difference in computer science classes. It does not help that the English language use the verb "to be" to indicate both classification (instances) and generalization (superclasses) (see [Fow00] p. 96). Usually, the difference is explained by contrasting "is an example of" and "is a kind of", and the instructor moves on to the next unit.

The issue becomes trickier when the instance *is* a class itself since both choices can appear valid. Sometimes the different characteristics of the relationships allow one possibility to be eliminated: generalization is transitive while classification is not; generalization can follow classification, but not the other way around. If applying transitivity leads to a nonsensical statement, or a desirable statement cannot be derived, the choice may become obvious. Furthermore, in stratified metamodels, the class' stratum is often predetermined and forces the developer's hand.

The issue becomes most interesting in unstratified metamodels. In these models, there is little to distinguish an instance from a subclass; often both choices are meaningful and will work in practice. For example, it is clear that *BinaryRelation* is a subclass of *Relation*, but should *Denote* be an instance or a subclass of *BinaryRelation*? Both arguments are expounded below.

From one point of view, *Denote* is an instance of *BinaryRelation* because it's clearly an example of a relation, not a whole class of them. It is only one set of pairs, not a collection of sets. Also, *BinaryRelation* is a useful class with actual relations as members since it can be used as the domain (or range) constraint for functions that operate on (or return) binary relations.

From the other point of view, *Denote* is just a subclass of *BinaryRelation*. In this approach, *BinaryRelation* is defined as the generic class of all pairs that describe binary relationships, and specific relations are merely refinements (subsets) of it. Having a domain or range of binary relations is only slightly trickier: all one needs to do is collect all subclasses of *BinaryRelation* into a new class, a simple query operation. It is also easy to restrict all binary relations to only contain pairs. A normal membership constraint can be applied to *BinaryRelation* to indicate that all members should be ordered nests of size 2. This is more difficult to do in the classification approach above, since the constraint needs to "reach down" across two layers of instantiation (from *BinaryRelation* to its relation members, and then to their pair members). A constraint this "deep" is not going to be easy (or even possible) to express using the constraint mechanisms usually available.

In short, both techniques have advantages and disadvantages. The classification technique was chosen somewhat arbitrarily for Braque's naïve upper ontology, mainly because classification appealed more to the author's intuition. However, the dilemma shows that both alternatives are viable, and raises the possibility that one of them is superfluous.

This raises an interesting question: is it ever necessary to create new metaclasses in an unstratified metamodel, or is single-layer subclassing sufficient for all models?

C.3. Indicators and Representation

The concepts of denotation and representation are very slippery, as evidenced by the large quantity of nuanced definitions provided in the dictionary (15 sense for “to represent” alone [Mil01]), the many pages devoted to the subject in philosophical papers, and the sometimes-heated disagreement when the notions surface in discussion groups. This makes it difficult to design a clean upper ontology that includes these ideas—as it must—and can integrate their various interpretations into a single framework. The approach taken in Section 3.3.7 is to loosely define some very general relations that can easily be mapped to while still providing the discrimination required by the end user.

It is interesting to consider how HTTP retrieval fits into this framework. The HTTP protocol states that a GET method retrieves a representation of the resource identified by the URI passed to the server, or in other words that “an entity corresponding to the requested resource is sent in the response” [FIG+99]. While everyone agrees that the entity returned is not the actual resource, opinions differ widely on what is allowed as a “representation”. Some contend that only a high fidelity digital copy of the resource should be returned, severely limiting the kinds of resources that can be identified by retrievable URIs. Others say that anything related to the resource can be returned, including descriptions, metadata, etc.

This is a situation where the web’s loose “do anything that makes sense to humans” principle conflicts with the semantic web’s requirement for precise semantics. Entertaining though it might be to see how this issue eventually gets resolved, a semantic web metamodel would be grossly incomplete if it did not provide some way to integrate information about the existing web into its ontological framework. Luckily, the NUO is up to the task thanks to its relaxed class definitions; these could of course be refined when the community settles on an answer.

For now, Figure C-2 shows how the response to a GET query on `http://www.uvic.ca/` could be represented in the Braque metamodel. The *Home* page document retrieved is an abstract concept that represents *UVic* the resource identified by the URI, without being directly related to the URI itself. The document itself is represented by its HTML source and may embed other abstract elements. One such is the *Image*, which is probably a representation of its own abstract resource with a different URI, though this is not shown in this diagram. We also take advantage of being able to refer to links and tag the *Home page Represent UVic* relationship with its retrieval timestamp.

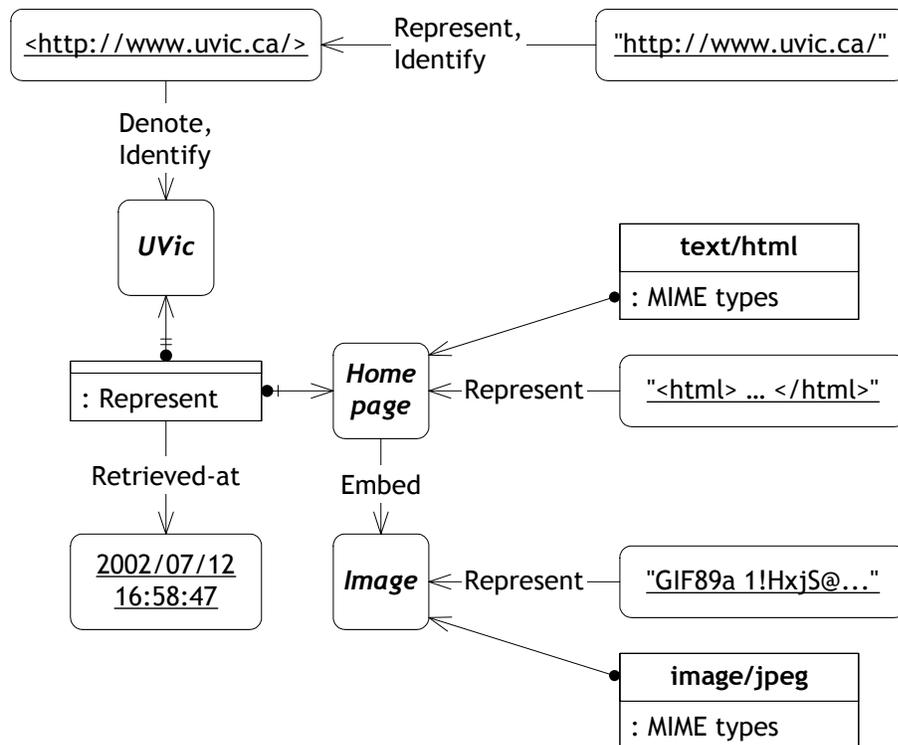


Figure C-2. Example representation of HTTP GET result

This is by no means a full mapping of all the information available when performing an HTTP retrieval. We could also model the request and reply themselves, and should consider mapping other header fields, e.g. “Content-Location”, which indicates the actual URI of the resource for which the representation was retrieved. Completing the mapping would require an in-depth un-

derstanding of the intricacies of HTTP and is a job for another day. However, this simple example shows that the *Represent* relation can be used appropriately when modeling web page retrieval.

Other interesting questions remain as well:

- Is *Represent* a reflexive relation?
- Are any of the *Indicate* relations transitive?

C.4. Punning on Classes

A class is a category of individuals that all share some common characteristics. The extent of a class is usually considered to be the set of its instances. However, it is sometimes tempting to reuse a class in other contexts, in effect subverting its identity for a related but incompatible purpose. This practice has somehow gained the name of “punning”, and it poses some difficult problems for meta-model integration. This section presents two examples of punning, then evaluates the practice and suggests strategies for dealing with it.

The first example of punning comes from RDF, specifically the current proposal for RDF datatypes [HMS02]. A datatype is defined to be both a class, whose extent is the set of values of that type, and a property (binary relation), whose extent is the mapping between values and literal representations thereof. This is permissible in RDF since the model theory [Hay02a] uses different functions to define the extent of classes and properties so the sets are kept separate. The advantage is that a given datatype name can be used both to indicate the type of a resource and to map a resource to its literal (Figure C-3).

```
_:age <rdf:value> _:ten .
_:ten <rdf:type> <xsd:decimal> .
_:ten <xsd:decimal> "10" .
```

Figure C-3. Datatype punning in RDF

Topic mappers are also incorrigible punsters: they often use the same subject as both a class and a role type. The justification proffered is that the concept of, say,

“student” outside the context of any given association is the same as that of the role of “student” in the association between a person and a university (Figure C-4). The punning does not affect the Topic Maps model (such as it is) since it does not explicitly deal with class or role type extent, but trips up formal models such as Braque or [AO+02].

```
[Piotr : student = "Piotr Kaminski"]
[UVic : university
  = "University of Victoria"
  @"http://www.uvic.ca"]
attends ( Piotr : student, UVic : university )
```

Figure C-4. Class and role punning in Topic Maps

In both cases, it seems intuitively wrong to use the same concept for two different purposes. While a class is sometimes considered to be a unary predicate, it is never interpreted as a binary one. In any case, most people would agree that unary and binary predicates (regardless of interpretation) are best kept separated. Furthermore, mixing up differing extents, even if acceptable for a model theory, is confusing when one wants to start making statements about the extents.⁴⁶

Punning also poses a particular problem in the Braque metamodel, since the extent of a class or relation is represented directly as its member elements. If a single concept is both a class and a relation, it will contain a mix of class instances and relation tuples, to the likely confusion of the user (not to mention the failure of some validity constraints). This is unacceptable, so barring a sudden ban on puns, what is a nice metamodel to do?

For punning to work at all, it must be obvious from context which meaning is intended. We could take advantage of this to split the single concept into two (or more) independent ones, each with a single interpretation and extent. This approach produces the cleanest mapping, but it will fail if ever the correct meaning is not obvious since the system would not know which concept to use. For ex-

⁴⁶ See Bob MacGregor’s testimony in his message to the RDF-Interest mailing list. <http://lists.w3.org/Archives/Public/www-rdf-interest/2002Jul/0129.html>

ample, in RDF, if a datatype is used as the subject of a statement, it is not clear whether the statement is about the class portion, the relation portion, or both. It works reasonably well for roles versus classes, though, and was adopted for the Topic Maps mapping.

Another approach would be to translate all but one of the extents into a relation other than membership. This lets us retain one central concept, but clearly distinguish between its different uses. For example, in a topic map, a role type subject would be linked to a role with an extra relationship rather than directly containing the reified membership pair. The problem there is that we'd be eschewing standard modeling techniques—thus confusing the user—and introducing extra levels of indirection into the model. The result might even become a partial object-to-semantic lift, whose elimination is a major goal of the Braque project.

A more promising avenue may be a re-examination of the semantics of classes and role playing. If we redefine class membership as “the instance plays the class role in a global context”, then assigning the class as a role in an association could be interpreted as “the member plays the class role in the context of the association”. This would require a change from the “flat” class extension to some more complicated scheme, and a complete rewrite of the NUO.

Finally, perhaps we should simply admit that the extent of a concept is context-dependent. The members of a nest would then be scoped, either globally, by functionality (e.g., RDF predicate), or by association, as would the type of the nest and hence the validation rules that apply to it. This approach may be the best yet, but would greatly complicate the simple membership semantics of Braque. It should be investigated with great care.

VITA

Surname: Kaminski

Given Names: Piotr

Place of birth: Gdańsk, Poland

Educational Institutions Attended:

University of Victoria	1998-2002
University of Waterloo	1992-1997

Degrees Awarded:

B. Math. (Honours Co-op)	University of Waterloo	1997
--------------------------	------------------------	------

Honours and Awards:

Canada Scholarship	1992-1997
Descartes Scholarship	1992-1997

Publications:

Applying Multi-dimensional Separation of Concerns to Software Visualization. Advanced Separation of Concerns Workshop, International Conference on Software Engineering, Toronto, Canada, 2001.

Using Hypersets to Represent Semistructured Data. Information Systems Modeling, Roznov pod Radhostem, Czech Republic, 2002.

UNIVERSITY OF VICTORIA PARTIAL COPYRIGHT LICENSE

I hereby grant the right to lend my thesis (or dissertation) to users of the University of Victoria Library, and to make single copies only for such users or in response to a request from the Library of any other university, or similar institution, on its behalf or for one of its users. I further agree that permission for extensive copying of this thesis for scholarly purposes may be granted by me or a member of the University designated by me. It is understood that copying or publication of this thesis for financial gain by the University of Victoria shall not be allowed without my written permission.

Title of Thesis/Dissertation:

Integrating Information on the Semantic Web
Using Partially Ordered Multi Hypersets

Author _____

Piotr Kaminski

September 26, 2002