Qualitative Reasoning and the Web Ontology Language

Jochem Liem
Supervisor: Dr. Bert Bredeweg

Doctoral Thesis
Artificial Intelligence - Knowledge Technology

Human Computer Studies Laboratory
Informatics Institute
Faculty of Science
University of Amsterdam

Kruislaan 419, Matrix I
1098 VA Amsterdam, The Netherlands

May 2005

Student Number: 9924728

Abstract

The desire to share and reuse knowledge has led to the establishment of a new knowledge representation language: the Web Ontology Language (OWL). The design choices made in OWL are properly documented, but their implications for AI research are not yet clear. This thesis explores the expressiveness of OWL by formalising the vocabulary and models used in qualitative reasoning (QR), and the applicability of OWL reasoners to solve QR problems. In addition, a parser has been developed to export the QR knowledge representation to an OWL representation. To formalise the OWL definitions of the QR vocabulary and models existing OWL patterns were used as much as possible. However, some new patterns had to be developed in order to cope with the QR vocabulary and models. Furthermore, some conceptual patterns which are impossible to define in OWL are revealed.

Acknowledgements

I would like to thank the following people for their help:

Bert Bredeweg for his guidance, suggestions and constructive criticism.

Jasper Uijlings for proofreading and suggesting improvements.

My parents and brother for their support.

My friends for forcing me to explain the purpose of my thesis.

Contents

1 Introduction **8**

2 Ontologies and Knowledge Engineering **10**
2.1 What is an Ontology? . . . 10
2.2 Ontologies in a Knowledge Engineering Context . . . 11
2.3 Categories of Ontologies and Data Models . . . 12
2.4 An Example Ontology and Data Model . . . 13
2.5 Conclusions . . . 13

3 The Web Ontology Language **15**
3.1 OWL File Preamble . . . 15
3.2 RDF Opening Tag and Namespaces . . . 16
3.3 Ontology Meta Data . . . 16
3.4 Classes and Subclasses . . . 16
3.5 Properties and Subproperties . . . 17
3.6 Characteristics of Properties . . . 17
3.7 Classes Defined by Property Restrictions . . . 18
3.8 Classes Defined Using Set Theory . . . 19
3.9 Individuals . . . 21
3.10 Classes Defined by Enumerations of Individuals . . . 22
3.11 The Three Species of OWL . . . 22
3.12 Conclusions . . . 22

4 Qualitative Reasoning **23**
4.1 Purpose of Qualitative Reasoning . . . 23
4.2 Qualitative Modelling . . . 23
4.2.1 Aggregates . . . 23
4.2.2 Structural . . . 24
4.2.3 Assumptions . . . 25
4.2.4 Behavioural . . . 25
4.3 Qualitative Model Simulation . . . 26
4.3.1 The Classification Task . . . 27
4.3.2 The Prediction Algorithm . . . 28
4.4 Conclusions . . . 29

5 Ontology of Qualitative Model Ingredients **30**
5.1 Ontology Validation and Knowledge Representation Parser . . . 30
5.2 A Hierarchy of Qualitative Model Ingredients . . . 30
5.2.1 The Hierarchy . . . 30
5.2.2 Reification of Relations . . . 32
5.2.3 Property Restrictions . . . 33
5.3 Qualitative Model Ingredient . . . 34
5.4 Structural . . . 34
5.4.1 Entities, Agents, and their Relations . . . 34
5.4.2 Configurations . . . 35
5.4.3 Attributes . . . 36
5.5 Assumption Types . . . 37
5.6 Behavioural . . . 38
5.6.1 Quantities, Magnitudes, Derivatives and Quantity Spaces . . . 38
5.6.2 Quantity Spaces and Qualitative Values . . . 39

5.6.3 Proportionalities and Influences . . . 42
5.6.4 Correspondences . . . 43
5.6.5 Inequalities . . . 44
5.6.6 PlusMin Relations . . . 47
5.7 Aggregate . . . 48
5.7.1 Model Fragments . . . 50
5.7.2 Scenarios . . . 50
5.8 Consequences of Aggregate Formalisation in Models on Reasoning . . . 51
5.9 Conclusions . . . 52

6 Conclusions, Discussion and Further Research 54
6.1 OWL Patterns . . . 54
6.2 Restriction Formalisation Problems . . . 55
6.3 Dealing with Complex Knowledge Structures . . . 55
6.4 Increasing Expressiveness . . . 55
6.5 OWL Reasoner Solving QR Problems . . . 56
6.6 The Qualitative Reasoning Vocabulary Revisited . . . 56
6.7 Towards Model Sharing and Reuse . . . 56

References 57

A OWL Patterns 60
A.1 Relation Reification Pattern 1 . . . 60
A.2 Relation Reification Pattern 2 . . . 60
A.3 Reificated Property Restrictions . . . 61
A.4 Property Values Patterns . . . 62
A.4.1 Property Values as a Set of Individuals . . . 62
A.4.2 Property Values as a Set of Classes . . . 62
A.4.3 A List as a Property Value . . . 63
A.4.4 An Ontological Ordering of Values . . . 64

B Qualitative Reasoning Vocabulary 64
B.1 The Qualitative Vocabulary Hierarchy . . . 64
B.2 Qualitative Model Ingredient Restrictions . . . 73
B.3 Entity and Agent Restrictions . . . 73
B.4 Configuration Restrictions . . . 73
B.4.1 Entity and Agent Configuration Restrictions . . . 74
B.5 Attribute Restrictions . . . 75
B.6 Quantity Restrictions . . . 75
B.7 Quantity Space Restrictions . . . 76
B.8 Causal Dependency Restrictions . . . 77
B.8.1 Quantity Causal Dependency Restrictions . . . 77
B.9 Correspondences . . . 77
B.10 Inequalities . . . 78
B.10.1 Inequalities Belonging to Points . . . 78
B.10.2 Inequalities Belonging to Magnitudes or Derivatives . . . 79
B.10.3 Inequalities belonging to Intervals . . . 80
B.10.4 hasValue Relations . . . 81
B.11 PlusMin Relations . . . 81
B.12 Aggregate . . . 84
B.12.1 Model Fragment . . . 84
B.12.2 Static Fragment . . . 84

B.12.3 Process Fragment 85
B.12.4 Agent Fragment 85
B.12.5 Scenario 86

C Example Qualitative Model Definitions **86**
C.1 Entity and Agent Definitions 86
C.2 Configuration Definitions 86
C.3 Assumptions 87
C.4 Attribute Definition 87
C.5 Quantity Space Definitions 87
C.6 Quantity Definitions 89
C.7 Model Fragment Definitions 89
C.8 Scenario Definitions 93

1 Introduction

The representation of knowledge in a form readable by both computers and humans is an important research topic in Artificial Intelligence (AI). There are three problems which have to be solved in order to make this kind of knowledge representation possible. Firstly, a language is needed in which knowledge can be described [5, 15, 18]. These languages are usually some kind of logical formalism. Secondly, a methodology has to be developed which describes how knowledge should be formalised [21, 37, 25, 27, 43, 48, 16, 19]. Until now, the development of knowledge representations is considered more of an art than a science. Finally, knowledge about subject areas has to be captured in so-called ontologies [22, 1]. These ontologies describe the vocabulary and facts of some universe of discourse, and the possible conclusions which may be deduced from them.

The languages developed thus far have not found wide adoption, preventing the establishment of a universal ontology language. However, the increase in the development of web services has awakened the desire to share and reuse knowledge and information. This led to World Wide Web Consortium (W3C) standards such as XML(S) and RDF(S), and more recently, the Web Ontology Language (OWL) [35, 45, 2, 39]. Even though OWL offers no new breakthroughs in knowledge representation, it does promise the capability to share and reuse knowledge, as it was specifically developed for the architecture of the world wide web. OWL has become one of the most widely adopted KR language to date. Universities and businesses are actively developing parsers, validators, editors and reasoners, making OWL a logical choice for the development of ontologies.

During the development of OWL, design choices have been to ensure the language is decidable and not too intricate to implement. Therefore, OWL does not have the expressiveness to formalise things such as, default values, arithmetic, string operations, or procedural attachments [49]. Another feature of OWL is that is has an open world assumption. The implications of these design choices on ontology development are still unclear. Therefore, the research question this thesis focusses on is: *"What are the consequences of the OWL design choices on the expressiveness of the language for AI applications?"*

To discover the problems and solutions associated with the utilisation of OWL, the models and vocabulary of Qualitative Reasoning (QR) [12, 13, 8, 9] will be formalised. There are five reasons this typical AI application is chosen. Firstly, qualitative reasoning predicts the behaviour of systems; a task rather different from the classification task OWL reasoners can solve. Secondly, the knowledge representation vocabulary used in QR models is elaborate and complex. For instance, qualitative models describe both the structure and the behavioural aspects of a system, parts of models may be reused in others, and the requirements for a correct model are restrictive in what kinds of ingredients may be connected. Thirdly, QR models describe their domains in a way non-experts find understandable, closely following the proposal of naive physics (except for the focus on implementation) [21]. This common-sense view of QR models on systems make them interesting to reuse. Fourthly, the OWL community proposes that using an ontology as an information model for design is a typical use case [24]. Since both modelling and design are synthesis tasks [43], the formalisation of the QR domain should be a typical application of OWL. This makes the problems discovered during the characterisation of the QR application area of interest to a large group of researchers. Finally, the effort to formalise the qualitative vocabulary and models in OWL creates a shareable format, which is one of the goals of the European NaturNet-Redime project[1]. In order to answer the general research question mentioned above this thesis focusses on the more specialised question: *"Is the Web Ontology Language expressive enough to formalise the QR vocabulary and models, and which of the QR problems can an OWL reasoner solve?".*

This thesis is organised into multiple chapters each answering some subquestions which are part of the solution of the main research problem. The thesis begins with chapter 2, which explains the concept ontology and its relation to knowledge representation. The subquestions posed are: *"What is an ontology?"*, and *"What are the uses of ontologies?"* Chapter 3 summarises the Web

[1] http://www.naturnet.org/

Ontology Language, and focusses on the question: *"What constructs are available in the Web Ontology Language?"* Chapter 4 provides an overview of qualitative reasoning. It provides answers to the questions: *"What is the purpose of qualitative modelling?"*, *"What does a QR model look like?"*, and *"What are the kind of tasks the qualitative engine solves?"* In chapter 5, the formalisations of the QR vocabulary and models in OWL are presented. It answers the questions: *"Is the Web Ontology Language expressive enough to capture the QR vocabulary and model formalisations?"*, *"Which existing general patterns can be used to solve formalisation problems during the development of the ontology, and which reusable solutions are discovered?"*, *"What conceptual patterns cannot be expressed in OWL?"*, and *"What QR problems can an OWL reasoner solve?"* Finally, chapter 6 summarises and discusses the conclusions of this thesis, and provides suggestions for further research.

2 Ontologies and Knowledge Engineering

This chapter establishes a definition of the word ontology. The first section describes the origins of the word, and how it was adopted by the AI community. The second and third section describe the use of ontologies from a knowledge engineering perspective. The final section provides an example ontology using the established terminology.

2.1 What is an Ontology?

The word ontology originates from a subfield of philosophy in which the nature of existence is studied. In this science, concepts are ordered into general categories, e.g. physical and abstract. The categories themselves are also concepts. These organisations of the most general concepts in existence are called philosophical ontologies [46], and are somewhat different from the ontologies used in knowledge engineering (section 2.2). As philosophical ontologies are developed in a top-down manner, i.e. from the most general categories to more specific ones, they are also referred to as top-ontologies.

Researchers in artificial intelligence, database management systems, and software engineering independently discovered the value of modelling concepts [44]. McCarthy was the one of the first to propose the formalisation of common sense as logical sentences [33]. He also recognised that building a program with common sense would require the formalisation of both general common sense knowledge (domain independent) and knowledge about particular situations (system and domain specific). So in order to reason about knowledge, the necessary concepts have to be introduced in an ontology [34]. This insight bridged the gap between the research in philosophical ontologies and knowledge representation. McCarthy's view to building a program with common sense inspired Hayes' famous work on naive physics, in which he proposed to formalise a large part of common sense knowledge about the physical world [21].

The most commonly cited definition of an ontology is the one given by Gruber: *"An ontology is an explicit specification of a conceptualization"* [20, 19]. A conceptualisation is an abstraction of reality; a simplified model in which the relevant concepts of some domain are expressed. Explicit means that the concepts and relations have to be fully and clearly defined. A specification is nothing more than an exact denotation.

Gruber's definition is considered too broad, as it could arguably encompass database schemas and catalogues. It has therefore been used as a basis for stricter definitions of the word ontology. Borst proposed the definition: *"An ontology is a formal specification of a shared conceptualization"*[4]. This definition accentuates that researchers must agree on the formalisation of the knowledge in ontologies, as without mutual consent the ontology would not be reusable. Studer et al. extended the definition to: *"An ontology is a formal, explicit specification of a shared conceptualisation"*[48].

Both Borst's and Studer's definition use the word formal to stress the fact that the ontology should be machine readable. The problem with this definition is that traditional philosophical ontologies would no longer fit the definition, as e.g. images of inheritance hierarchies are not formal in the sense meant by Borst and Studer. Instead of the focus on machine readability, it is far more important that ontologies provide a strict interpretation of the concepts they define, and restrictions in how they may be used. These are issues which have to be coped with during the development of both philosophical ontologies and ontologies in knowledge engineering contexts. For example, stating that objects are either abstract or physical, means that there are no things which are both. If a concept is found which belongs to both categories, the ontology is inconsistent and has to be adapted. The interpretation of, and the restrictions on the ontology provide the semantics of the ontology.

The definition which will be used throughout this thesis is the following: *"An ontology is an explicit specification of a shared conceptualisation which has a strict interpretation and restricts the use of introduced concepts."* Ontologies such as those described by Sowa [46] still fit this definition,

while relatively unstructured knowledge systems such as catalogues, dictionaries and thesauri are excluded.

2.2 Ontologies in a Knowledge Engineering Context

The use of ontologies in knowledge engineering is different from the purpose of ontologies in philosophy. In philosophy the main goal is the creation of ontologies to order the things that exist. In contrast, knowledge engineering ontologies are created with the purpose of constraining models of the world and indicating possible deductions (inferences).

To be able to use an ontology in a knowledge system, it has to be formalised using some knowledge representation language. The nature of a knowledge representation can best be understood by the five distinct roles it plays [11]. In the first place, a knowledge representation is a surrogate for the world. It is used to reason about actions on the world instead of determining their result by acting on the world. Secondly, a knowledge representation is a set of ontological commitments, as it determines the point of view from which the world is seen. Thirdly, a knowledge representation is a fragmentary theory of intelligent reasoning. The representation determines what things are rational to infer (sanctioned inferences), and sometimes what things should be inferred (recommended inferences). These possible inferences reveal a lot about the underlying hypotheses about rationality. The theory is fragmentary in the sense that the formalisation is only part of the idea that inspired it, and the idea in turn is only part of what is commonly considered to be intelligent reasoning. Fourthly, a knowledge representation is a medium for efficient computation, as it determines how difficult it is to make certain inferences. Finally, a knowledge representation is a medium for human expression. Therefore, it should be as effortless as possible for users to make expressions about the world.

The knowledge representation languages used to develop ontologies are called ontology languages. These languages formalise knowledge in the form of networks of nodes and binary relations, called semantic nets. These networks are formalised in logic using binary predicates, while in earlier KR predicates of arbitrary length could be used. One of the first language using this paradigm was KL-ONE, but an important inference was proven to be undecidable in this language [42]. The lessons learned from KL-ONE inspired researchers to work on so called description logics (DL). These languages maintain the concept-centred perspective of semantic nets, but usually describe them using decidable parts of first order logic (FOL). Other important languages which focussed more on the reuse and sharing of ontologies are the Knowledge Interchange Format (KIF) [15], and the Ontolingua Language [18], which was based on KIF. As most ontology languages are not widely used, knowledge sharing and reuse is hard.

Besides concepts, which describe general concepts, ontology languages can also be used to describe instances of classes. The concepts describe sets of possible objects, for example the general concept tree. Instances, however, describe actual objects, such as the tree in the centre of the town square. From the perspective of a system, an instance of a concept is said to exist if it is represented. Instances of concepts, or individuals as they are also called, correspond to specific entities in the world, and concepts to abstract concepts. Therefore an individual can change, but a concept cannot. A model in which the existing instances and relations are specified is commonly called a knowledge base.

The use of the term knowledge base is ambiguous. Some researchers state that a knowledge base contains rules, and is separated from a domain schema [43] (a special kind of ontology, see section 2.3). Others researchers maintain that a knowledge base includes an ontology, and do not even mention rules [47]. According to Newell: *"Knowledge is to be characterised entirely functionally, in terms of what it does, not structurally, in terms of physical objects with particular properties and relations"* [36]. One could argue that a knowledge base should therefore also contain the reasoning, as the results of reasoning show that a system is knowledgeable. All these interpretations of the word knowledge base result in a concept which is too ambiguous to be used. Instead of the word knowledge base, the word *data model* is used in this thesis to refer to a set of instances and their

relations. The word data is chosen, as a data model has no interpretation without its corresponding ontology. A data model contains neither an ontology nor rules. The word data model can also be used to refer to the specific information an agent has about a situation.

2.3 Categories of Ontologies and Data Models

With respect to the subject of the conceptualisation four different categories of ontologies (figure 1) can be distinguished [25].

- Representation ontologies correspond to the ontological commitments knowledge representation formalisms make. For example, most knowledge representation languages assume the existence of concepts, relations and instances. These ontological commitments have to be domain independent, as otherwise the formalism would be unusable for some domains. The primitives in representational ontologies are used as a framework to build generic ontologies.

- Generic ontologies extend the ontological commitments made by representational ontologies. Similarly to representational ontologies, the defined concepts and relations are domain independent. Concepts which are commonly found in generic ontologies are states, processes, actions, agents, etc. These concepts are defined as specialisations of the concepts defined in the representational ontologies.

- Domain ontologies further extend the ontological commitments made by generic ontologies. Domain ontologies are used to formalise conceptualisations which are specific for particular areas of discourse. As such, the domain ontology defines the vocabulary used in a specific domain. The distinction between generic and domain ontologies is partly vague, because if a domain is chosen broad enough it can be considered generic. Domain ontologies are also referred to as domain schemas [43].

- Application ontologies are used to provide storage for application specific information, which is essential for particular applications. They usually extend either the generic or domain ontologies using primitives from the representational language. As the information in application ontologies is highly specialised for one application in one domain, it is usually not reusable. As application specific information unnecessarily complicates an ontology, its use should be minimised as much as possible.

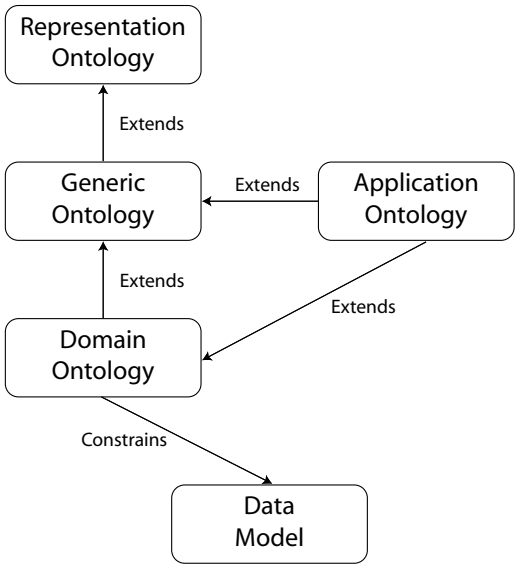

Figure 1: The different kinds of ontologies and the data model.

In figure 1, the discussed relations between the different kinds of ontologies and the data model are shown. The domain ontology describes knowledge about the domain, but no knowledge about particular situations in that domain. The latter information is formalised in the data model, and is embodied by instances of concepts and relations from the domain ontology. Unfortunately, some application specific information usually has to be stored. This information is also stored in the data model using the conceptualisation defined in the application ontology. The domain ontology constrains the structure of the data model, e.g. if abstract and physical concepts are defined, an individual in the data model cannot be an instance of both. The constraints exercised by ontologies and on the data model and the inferences they sanction are the semantics of the ontology.

2.4 An Example Ontology and Data Model

In figure 2 an example ontology and data model are shown. Consider a knowledge representation language which provides concepts and relations as ingredients. These ingredients are the representational ontology with which specialised ontologies can be constructed. In the generic ontology the concepts abstract and physical are defined. These notions are domain independent and can therefore be reused in other ontologies.

In the domain ontology the concepts of a toy contained liquid ontology are shown. There are containers, such as coffee cups and barrels, and liquids, such as oil and coffee, of which the instances are physical. Furthermore quantities, which are abstract entities, such as volume are defined. The relations has quantity and contains are also created, as liquids have a volume, and containers can contain something.

Consider an application used to visualise the containers and liquids. An ontology should provide a place to store the position where the containers and liquids have to be rendered on a screen. Therefore, in the application ontology the relation has position on screen, and the abstract concept position on screen are defined. As these concepts are only useful for the specific task of visualising data models of contained liquids, this part of the ontology is probably not reusable.

Finally in the data model a specific situation is described using instances of the concepts defined in the ontology. In this case, there is barrel containing 50 litres of oil, and a coffee cup holding 50ml of coffee. As all these things have to be visualised using a tool, their positions on screen are stored. The application might make it possible to move the objects, and store the new positions in the data model.

Although this example does not make use of any interesting reasoning, ontologies are used to make reasoners reusable. By defining a vocabulary a reasoner can understand an ontology and data model, and provide the possible inferences. For example, a relation can be defined to be transitive, or the inverse of another relation. These standard properties of relations have a specific interpretation, which reasoners implement. These definitions can be reused in ontologies, and users of those ontologies can reuse the reasoners.

2.5 Conclusions

This section provided the answers to the questions *"What is an ontology?"* and *"What are the uses of ontologies?"*. An ontology has been defined as an explicit specification of a shared conceptualisation which has a strict interpretation and restricts the use of introduced concepts. Ontologies are used to categorise the concepts in a domain, and define the vocabulary. Furthermore in a knowledge engineering context, they are used to restrict the data model and indicate possible inferences.

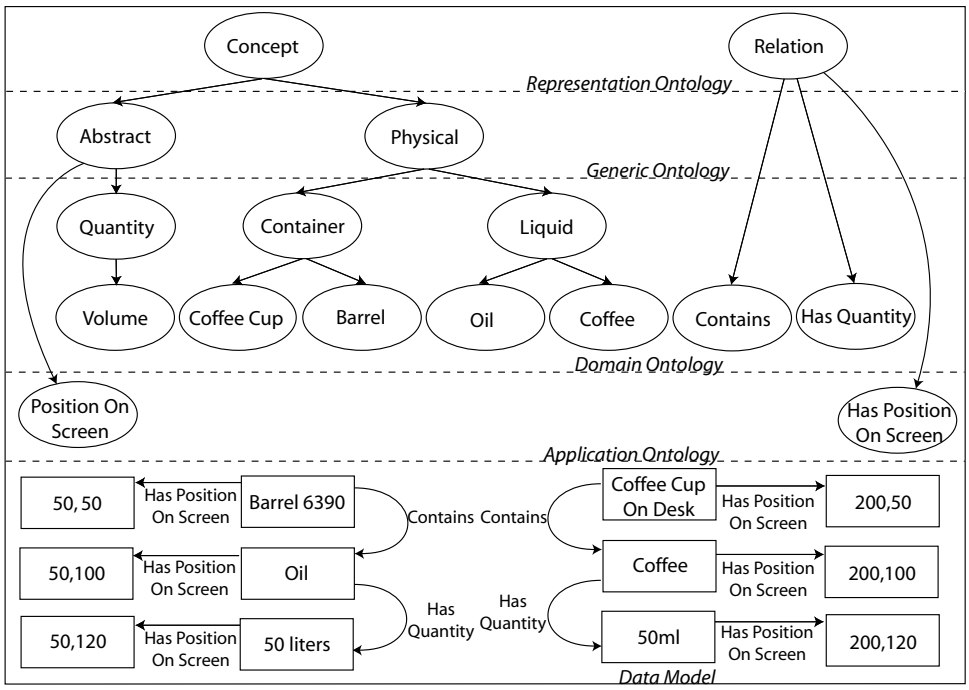

Figure 2: An example ontology and data model.

3 The Web Ontology Language

For the purpose of this project, the Web Ontology Language (OWL) [45, 2, 39] was chosen as the knowledge representation language to formalise qualitative reasoning vocabulary and models in. To be able to ascertain and describe the problems associated with the formalisation of AI ontologies, this section describes the KR constructs available in OWL.

OWL is part of the semantic web vision, and recently became a W3 recommendation. It has a large user base, and a lot of tools are being developed for its use[2]. OWL is built upon other W3C recommendations, as is shown in figure 3. The Extensible Markup Language (XML) [7] provides the structure for the documents, but provides no semantics. XML-Schema is a language used to restrict the structure of XML documents and provides data types. Namespaces (NS) provide the means to refer to elements and attributes used in XML documents by associating them with namespaces identified by URIs [6]. The Resource Description Framework (RDF) [3, 30] is a data model inspired by semantic nets (section 2.2). It is possible to formalise this data model in XML, using resources to describe objects and relations between them. RDF provides some simple semantics. RDF Schema [10, 32, 23, 17] extends RDF by introducing constructs to describe classes and properties, and provides the semantics for inheritance hierarchies. The DAML+OIL language was based on RDF Schema. From DAML+OIL the OWL language was derived.

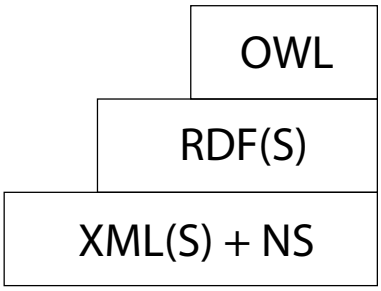

Figure 3: The Layering of W3C Standards.

3.1 OWL File Preamble

As every OWL document is a well-formed XML document, the first line of the OWL file has to be a XML declaration. In the XML declaration the version of XML being used is specified, and optionally in which character encoding the file is saved.

```
<?xml version="1.0"?>
```

After the XML declaration follows a document type declaration (DOCTYPE) in which entities (`&entityname;`) can be defined. These entities can then be used throughout the document as abbreviations of uniform resource identifiers (URI's).

```
<!DOCTYPE rdf:RDF [
    <!ENTITY animal             "http://phyro.mine.nu/animal2.xml#" >
    <!ENTITY colour             "http://phyro.mine.nu/colour.xml#" >
    <!ENTITY owl                "http://www.w3.org/2002/07/owl#" >
    <!ENTITY xsd                "http://www.w3.org/2001/XMLSchema#" >
] >
```

The XML declaration and the document type declaration are the only statements that do not have both an opening tag (`<tagname>`) and a closing tag (`</tagname>`). If a tag does not have any elements nested inside it (`<tagname></tagname>`), the empty element syntax may be used (`<tagname/>`).

[2]For a list of tools, projects and applications see: http://www.w3.org/2004/OWL/#projects

3.2 RDF Opening Tag and Namespaces

The first opening tag in the OWL file is `<rdf:RDF>`, which is closed at the end of the file by `</rdf:RDF>`. The attributes of the opening tag define some XML namespaces. XML namespaces define collections of names, which are identified by an URI. These namespaces are used in XML documents as element types and attribute names, which are defined at the specified URI [6]. Complete URI's in the opening tag can be exchanged for entities defined in the doctype.

```
<rdf:RDF
    xmlns               ="&animal;"
    xmlns:animal        ="&animal;"
    xml:base            ="&animal;"
    xmlns:colour        ="&colour;"
    xmlns:owl           ="http://www.w3.org/2002/07/owl#"
    xmlns:rdf           ="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:rdfs          ="http://www.w3.org/2000/01/rdf-schema#"
    xmlns:xsd           ="http://www.w3.org/2001/XMLSchema#">

<!-- The rest of the OWL file -->

</rdf:RDF>
```

The first namespace defines the default namespace, meaning that unprefixed names refer to the current ontology. The second specifies that the current ontology can be referred to using the prefix `animal:`. The third defines the base URL for the ontology. The fourth specifies the namespace of the imported (see below) colour ontology. The fifth through eight namespace declarations state that constructs prefixed with `owl:`, `rdf:`, `rdfs:` and `xsd:` refer to concepts defined in their corresponding namespaces.

3.3 Ontology Meta Data

For purposes of searching for ontologies it is desirable to be able to store meta data about ontologies. This kind of information can be encapsulated inside `owl:Ontology` tags. `owl:label` is used to provide a name for the ontology, but is also (just as `rdfs:comment`) used throughout the ontology to annotate classes and relations. `rdfs:comment` can be used to provide a human readable comment about the ontology. Using the attribute `xml:lang` labels and comments can be provided in multiple languages. `owl:versionInfo` can be used to state the current version of the ontology. `owl:priorVersion` provides an URL to the prior version of the ontology. Finally `owl:imports` is used to import other ontologies. Note that the imported ontology is also mentioned in the doctype and `rdf:RDF` tag.

```
<owl:Ontology rdf:about="">
    <rdfs:comment>An example ontology about animals</rdfs:comment>
    <owl:versionInfo>Version 2</owl:versionInfo>
    <owl:priorVersion rdf:resource="http://phyro.mine.nu/animal1.xml"/>
    <rdfs:label xml:lang="en">Animal Ontology</rdfs:label>
    <rdfs:label xml:lang="nl">Dieren Ontologie</rdfs:label>
    <owl:imports rdf:resource="http://phyro.mine.nu/colour.xml"/>
</owl:Ontology>
```

If the capability to store more meta data about the ontology is desired, the Dublin Core Meta Data standard [26] OWL-file can be imported. This meta data ontology defines concepts such as creator, subject, publisher, contributor, type, format, language, etc.

3.4 Classes and Subclasses

The most simple class which can be defined using owl is a class with just a name. To give the class its name `rdf:ID` is used. Implicitly every class is a subclass of `owl:Thing`, which therefore becomes

the root node of every class hierarchy. The empty class is defined as `owl:Nothing`. Subclasses are formalised using `rdfs:subClassOf`. To refer to classes or properties named using `rdf:ID`, `rdf:resource` is used.

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

<owl:Class rdf:ID="LivingBeing">
    <rdfs:subClassOf rdf:resource="#PhysicalThing"/>
</owl:Class>

<owl:Class rdf:ID="Animal">
    <rdfs:subClassOf rdf:resource="#LivingBeing"/>
</owl:Class>
```

3.5 Properties and Subproperties

Relations, called properties in OWL, are defined using `owl:ObjectProperty`. In the definition of properties `rdfs:domain` and `rdfs:range` may be specified. The former specifies that the owner of the relation is of the specified class. The latter specifies that the target of the relation is of the specified class. Note that these are not restrictions, but possible inferences. An OWL reasoner may infer the type of the owner and target of the relation if these characteristics are defined. Like classes, properties are ordered in a hierarchy. To state that a property is a subproperty `rdfs:subPropertyOf` is used. `owl:inverseOf` is utilised to specify that a property is the inverse of another property. From property K, its inverse L and the relation $\mathsf{K}(\mathsf{x},\mathsf{y})$, it is valid to infer $\mathsf{L}(\mathsf{y},\mathsf{x})$. `owl:equivalentProperty` is used to state that two properties have the same values, which does not mean that the properties are the same. Relations between instances of classes and RDF literals or XML Schema data types are specified using `owl:DatatypeProperty`. Commonly used XML Schema data types are `xsd:nonNegativeInteger` and `xsd:string`.

```
<owl:ObjectProperty rdf:ID="consumes">
    <rdfs:domain rdf:resource="#Animal"/>
    <rdfs:range  rdf:resource="#PhysicalThing"/>
</owl:ObjectProperty>

<owl:ObjectProperty rdf:ID="isConsumedBy">
    <owl:inverseOf rdf:resource="#consumes"/>
</owl:ObjectProperty>

<owl:ObjectProperty rdf:ID="eats">
    <rdfs:subPropertyOf rdf:resource="#consumes"/>
    <rdfs:domain rdf:resource="#Animal"/>
    <rdfs:range  rdf:resource="#PhysicalThing"/>
</owl:ObjectProperty>

<owl:DatatypeProperty rdf:ID="name">
    <rdfs:domain rdf:resource="#Dog"/>
    <rdfs:range  rdf:resource="&xsd;string"/>
</owl:DatatypeProperty>

<owl:ObjectProperty rdf:ID="hasColor">
    <rdfs:domain rdf:resource="#PhysicalThing"/>
    <rdfs:range  rdf:resource="&colour;Colour"/>
</owl:ObjectProperty>
```

3.6 Characteristics of Properties

OWL provides the possibility to add characteristics to properties which allow additional inferences. A relation can be defined to be transitive by stating that the property is of type `&owl;TransitiveProperty`. Consider a transitive relation R. From the relations $\mathsf{R}(\mathsf{a},\mathsf{b})$ and $\mathsf{R}(\mathsf{b},\mathsf{c})$

the relation R(a,c) may be inferred. Examples of relations which are transitive are equalities, inequalities and subset relations.

```
<owl:ObjectProperty rdf:ID="isSubsetOf">
    <rdf:type rdf:resource="&owl;TransitiveProperty"/>
    <rdfs:domain rdf:resource="#Set"/>
    <rdfs:range rdf:resource="#Set"/>
</owl:ObjectProperty>
```

Symmetric relations are specified by defining the property as having type `&owl;SymmetricProperty`. Consider a symmetric relation P. From P(x,y) the relation P(y,x) may be inferred. Equality, adjacency, and being married to are symmetric relations.

```
<owl:ObjectProperty rdf:ID="isMarriedTo">
    <rdf:type rdf:resource="&owl;SymmetricProperty"/>
    <rdfs:domain rdf:resource="#Person"/>
    <rdfs:range rdf:resource="#Person"/>
</owl:ObjectProperty>
```

Functional properties specify that there exists only a single individual which is the value of the property. These kind of properties can be defined by specifying that the property is of type `&owl;FunctionalProperty`. From a functional property Q and the relations Q(x,y1) and Q(x,y2), you may infer that y1=y2. Examples of functional properties are: hasFather, hasBirthDate, hasGender.

```
<owl:ObjectProperty rdf:ID="hasFather">
    <rdf:type rdf:resource="&owl;FunctionalProperty"/>
    <rdfs:domain rdf:resource="#Person"/>
    <rdfs:range rdf:resource="#Person"/>
</owl:ObjectProperty>
```

Inverse functional properties specify that the value of a relation is unique for individuals having that property. Stating that the property is of type `&owl;InverseFunctionalProperty` means that the relations is an inverse functional property. From an inverse functional property R and the relations R(x1,y) and R(x2,y), you may infer that x1=x2. Examples of inverse functional relations are: hasSocialSecurityNumber, hasDollarSerialNumber, hasStudentNumber.

```
<owl:ObjectProperty rdf:ID="hasSocialSecurityNumber">
    <rdf:type rdf:resource="&owl;InverseFunctionalProperty"/>
    <rdfs:domain rdf:resource="#Person"/>
    <rdfs:range rdf:resource="#Person"/>
</owl:ObjectProperty>
```

3.7 Classes Defined by Property Restrictions

Restrictions always make use of the `owl:onProperty` construct, specifying the property to which the restriction applies. There should always be a second ingredient used in restrictions, which state the kind of restriction that applies to the property. There are several of these kinds of restrictions: `owl:hasValue` specifies the instance of a class (see section 3.9 for more about individuals) or data type that must be the value of the property. `owl:allValuesFrom` states that the values of the property must be of a certain class, and acts as a universal quantifier. `owl:someValuesFrom` specifies that some of the values of the property must be of a certain class, and acts as a existential quantifier. `owl:cardinality, owl:minCardinality` and `maxCardinality`, specify the exact, minimum and maximum number of values a property must have. All constructs on the same indentation level should be read as having a conjunction in between them. So multiple restrictions have to be written as multiple restriction blocks.

In the next example, the concept polar bear is defined to be a subclass of the concept bear and an anonymous class with certain restrictions. In this case a polar bear must have a white colour. It is perfectly valid to define a class using solely restrictions.

```
<owl:Class rdf:ID="PolarBear">
    <rdfs:subClassOf rdf:resource="#Bear"/>
    <rdfs:subClassOf>
        <owl:Restriction>
            <owl:onProperty rdf:resource="#hasColor"/>
            <owl:hasValue rdf:resource="&colour;White"/>
        </owl:Restriction>
    </rdfs:subClassOf>
</owl:Class>
```

The restrictions which belong to a class do not need to be stated directly. It is possible to extend the definition of a class defined in a file somewhere else on the web. To extend a class the keyword `rdf:about` is used to refer to the class.

```
<owl:Class rdf:ID="PolarBear">
    <rdfs:subClassOf rdf:resource="#Bear"/>
</owl:Class>

<owl:Class rdf:about="#PolarBear">
    <rdfs:subClassOf>
        <owl:Restriction>
            <owl:onProperty rdf:resource="#hasColor"/>
            <owl:hasValue rdf:resource="&colour;White"/>
        </owl:Restriction>
    </rdfs:subClassOf>
</owl:Class>
```

The reason classes can be extended is because a subclass definition in OWL states necessary constraints, but not sufficient. This means that in the previous example you can infer that a polar bear must be white, but not that a bear, which is white is a polar bear. It is possible to state necessary and sufficient restrictions for a class by using `owl:equivalentClass` or `owl:intersectionOf` (section 3.8). Being an instance of a specific class implies that its necessary conditions apply, but an instance fulfilling the necessary and sufficient conditions of a class must be a member of that class.

```
<owl:Class rdf:ID="#PinkThing">
    <owl:equivalentClass>
        <owl:Restriction>
            <owl:onProperty rdf:resource="#hasColor" />
            <owl:hasValue   rdf:resource="&colour;Pink" />
        </owl:Restriction>
    </owl:equivalentClass>
</owl:Class>
```

3.8 Classes Defined Using Set Theory

Classes can be considered to be sets of individuals. By using set operators on these sets of individuals, other classes can be defined. Like with subclass definitions both classes and anonymous classes can be used.

To define a class as the intersection of other classes and restrictions `owl:intersectionOf` is used. This characteristic requires a `rdf:parseType="Collection"` attribute, which indicates that a list of classes follows. The listing specifies the classes to which the instance must belong and the restrictions it must abide to. Note that `rdf:about` must be used to refer to classes in the intersection, as the definition adds to the definition of the class.

```
<owl:Class rdf:ID="Herbivore">
    <owl:intersectionOf rdf:parseType="Collection">
        <owl:Class rdf:about="#Animal"/>
        <owl:Restriction>
```

```
            <owl:onProperty rdf:resource="#eats"/>
            <owl:allValuesFrom rdf:resource="#Plant"/>
        </owl:Restriction>
        <owl:Restriction>
            <owl:onProperty rdf:resource="#eats"/>
            <owl:minCardinality rdf:datatype="&xsd;nonNegativeInteger">1
            </owl:minCardinality>
        </owl:Restriction>
    </owl:intersectionOf>
</owl:Class>

<owl:Class rdf:ID="Omnivore">
    <owl:intersectionOf rdf:parseType="Collection">
        <owl:Class rdf:about="#Animal"/>
        <owl:Restriction>
            <owl:onProperty rdf:resource="#eats"/>
            <owl:someValuesFrom rdf:resource="#Animal"/>
        </owl:Restriction>
        <owl:Restriction>
            <owl:onProperty rdf:resource="#eats"/>
            <owl:someValuesFrom rdf:resource="#Plant"/>
        </owl:Restriction>
    </owl:intersectionOf>
</owl:Class>
```

A class definition using a union of classes is very similar to the class definition using an intersection. The difference is that `owl:unionOf` is used. To be a member of the defined class, an individual must be a member of one of the classes in the union.

```
<owl:Class rdf:about="#Vertebrate">
    <owl:unionOf rdf:parseType="Collection">
        <owl:Class rdf:about="#Fish"/>
        <owl:Class rdf:about="#Amphibian"/>
        <owl:Class rdf:about="#Reptile"/>
        <owl:Class rdf:about="#Bird"/>
        <owl:Class rdf:about="#Mammal"/>
    </owl:unionOf>
</owl:Class>

<owl:Class rdf:about="#PhysicalThing">
    <owl:unionOf rdf:parseType="Collection">
        <owl:Class rdf:about="#LivingBeing"/>
        <owl:Class rdf:about="#NonLivingThing"/>
    </owl:unionOf>
</owl:Class>
```

To indicate that the individuals of a class are exactly all the members which do not belong to another class, the `owl:complementOf` construct is used. This construct can also be used in combination with restrictions, to specify all the individuals which do not fulfil the restriction.

```
<owl:Class rdf:ID="NonLivingThing">
    <rdfs:subClassOf rdf:resource="#PhysicalThing"/>
    <owl:complementOf rdf:resource="#LivingBeing"/>
</owl:Class>

<owl:Class rdf:ID="Vegetarian">
    <owl:intersectionOf rdf:parseType="Collection">
        <owl:Class rdf:about="#Animal"/>
        <owl:complementOf>
            <owl:Restriction>
                <owl:onProperty rdf:resource="#eats"/>
                <owl:hasValue rdf:resource="#Animal"/>
```

```
            </owl:Restriction>
        </owl:complementOf>
    </owl:intersectionOf>
</owl:Class>
```

To specify that members of a class cannot be simultaneously members of another class the `owl:disjointWith` construct is used.

```
<owl:Class rdf:about="#Animal">
    <owl:disjointWith rdf:resource="#Plant" />
    <owl:disjointWith rdf:resource="#Bacteria" />
    <owl:disjointWith rdf:resource="#Fungus" />
    <owl:disjointWith rdf:resource="#Virus" />
</owl:Class>
```

As mentioned in section 3.7 using boolean combinations (and `owl:equivalentClass`) it is possible to state necessary and sufficient conditions. In the next example a polar bear is defined as the intersection of the class bear and things which are white. In contrast to the polar bear definition in 3.7 it is now valid to infer from an individual which is a bear and is white that it is a polar bear.

```
<owl:Class rdf:ID="PolarBear">
    <owl:intersectionOf rdf:parseType="Collection">
        <owl:Class rdf:resource="#Bear"/>
        <owl:Restriction>
            <owl:onProperty rdf:resource="#hasColor"/>
            <owl:owl:hasValue rdf:resource="&colour;White/>
        </owl:Restriction>
    </owl:intersection>
</owl:Class>
```

3.9 Individuals

The definition of classes sometimes depends the existence of individuals. For example classes can be defined as things with properties having specific values, or as the enumeration of the individuals which belong to the class. Therefore, *"in OWL the term ontology has been broadened to include instance data (...)"* [45].

Instances of classes and instances of properties are introduced using the names of those classes and properties. Another way is to introduce an `owl:Thing` and specify its type using `rdf:type`.

```
<Bear rdf:ID="JohnnyBear">
    <eats rdf:resource="#MikeGoldFish"/>
    <hasColor rdf:resource="&colour;White"/>
</Bear>

<owl:Thing rdf:ID="WilliamBear">
    <rdf:type rdf:resource="#Bear"/>
    <eats rdf:resource="#MikeGoldFish"/>
    <hasColor rdf:resource="&colour;White"/>
</owl:Thing>
```

As OWL does not have a unique name assumption, the two bears in the previous example are not necessarily different. It is possible to say that the two are the same using `owl:sameAs`.

```
<Bear rdf:about="#WilliamBear">
    <owl:sameAs rdf:resource="#JohnnyBear" />
</Bear>
```

It is also possible to say that the two bears are different using `owl:differentFrom`. It would be rather unwieldy to specify for each individual that it is different from the others. Therefore, it is possible to state that all individuals in a set are distinct using `owl:allDifferent` in combination with `owl:distinctMembers`.

```
<Bear rdf:about="#WilliamBear">
    <owl:differentFrom rdf:resource="#JohnnyBear" />
</Bear>

<owl:allDifferent>
    <owl:distinctMembers rdf:parseType="Collection">
        <Bear rdf:about="WilliamBear" />
        <Bear rdf:about="JohnnyBear" />
    </owl:distinctMembers>
</owl:allDifferent>
```

3.10 Classes Defined by Enumerations of Individuals

OWL provides the possibility to define a class by specifying its members. Such an enumeration is done using the `owl:oneOf` construct.

```
<owl:Class ID="Gender">
    <owl:oneOf rdf:parseType="Collection">
        <owl:Thing rdf:about="#Male"/>
        <owl:Thing rdf:about="#Female"/>
    </owl:oneOf>
</owl:Class>
```

3.11 The Three Species of OWL

OWL consists of three increasingly expressive sublanguages. Every ontology written in a less expressive sublanguage is a legal ontology in the more expressive sublanguages.

OWL Lite supports only the values 0 and 1 for the cardinality constraints `owl:minCardinality` and `owl:maxCardinality`.

OWL DL is a description logic, as OWL DL can be mapped to the $\mathcal{SHIQ}$ logic [29]. The idea is that OWL DL maintains the maximum expressiveness while maintaining computational completeness (all inferences are guaranteed to be computed), and decidability (the inferences will be computed in finite time). This is an excellent example of the tradeoff between expressiveness and decidability [31]. The distinctive restriction in OWL DL is type separation. It is forbidden to treat a a class as an instance, and a property as an instance of a class. A disadvantage of OWL Lite and OWL DL is that backwards compatibility with RDF is lost. This means that not every valid RDF file is a valid OWL Lite or DL file.

OWL Full has maximum expressiveness and is backwards compatible with RDF, but there are no computational guarantees. There is no type separation restriction, so classes can be treated as individuals, and properties as instances of properties. Data type properties can be given the property characteristic `owl:InverseFunctionalProperty`.

3.12 Conclusions

This section has summarised the Web Ontology Language and answered the question: *"What constructs are available in the Web Ontology Language?"* Constructs have been presented to define class and property hierarchies, add characteristics allowing inferences to properties, add restrictions to classes, define classes using set theory, and stating necessary and sufficient conditions.

4 Qualitative Reasoning

This section discusses the purpose of qualitative reasoning, how qualitative models are structured, and how the qualitative simulation works.

4.1 Purpose of Qualitative Reasoning

The aim of qualitative modelling and reasoning [12, 13, 8, 9] is to build a model from which the behaviour of a system can be derived. The main distinction between qualitative models and mathematical models is that QR models require no numerical data. Instead, changeable properties of systems are described as its the relevant points and intervals. For example, the temperature of a liquid can be formalised as {absoluteZero,Negative,FreezingPoint,Positive,BoilingsPoint,AboveBoiling}. The formalisation of values as point and intervals is particularly advantageous for domains, such as ecology, in which it is difficult to obtain specific numerical data. For experts in these fields, qualitative modelling provides a good alternative to make their knowledge explicit. An example of the application of qualitative modelling in ecology is the testing of the succession hypothesis of the Brazilian Cerrado forest [41].

In order to explain another feature of qualitative modelling the formalisation of the behaviour resulting from the simulation has to be explained. There are two types of behavioural descriptions which can be produced by the qualitative simulator. The first is called a full envisionment, which is a state graph containing all the possible situations and consistent transitions which can occur in a system. The second is an attainable envisionment, which is also a state graph, but is based on a predetermined begin situation of the system. In figure 4 a full envisionment of a U-tube system is shown. Each fluid container within this system can either be full, have contents, or be empty. From a correct model all the possible states and valid transitions are determined. An attainable envisionment starts from a specified begin situation, which is called a scenario, and consists of all the states and transitions reachable from that scenario. A few examples of attainable envisionments are the following subsets in figure 4: {2,7,9}, {4,8,9} and {11}.

An advantage of qualitative modelling over mathematical modelling is that the causal dependencies between quantities are made explicit. Next to the ability to predict the behaviour of the system, these causal dependencies make it possible to provide a causal explanation of why the system behaves in a particular way. These features of qualitative simulation provide the opportunity for hypothesis testing and learning. To be able to model a system there must be some notion about the structure and causal relations of the system and the behaviour it exhibits. More often then not, the simulator generates a state graph which is inconsistent with the anticipated behaviour. In those cases either the model is incorrect, or the system displays behaviour which was not anticipated. In the first situation, the modeller learns that his ideas about the structure of, or the processes in the system were incorrect. In the second, he ascertains that the system exhibits behaviour which was not predicted.

4.2 Qualitative Modelling

GARP [8] is a domain independent reasoning engine that implements a compositional modelling approach to qualitative simulation. GARP makes use of concepts defined in QPT [13], compositional modelling [14], and the confluences based approach [12]. GARP models can be either written using a text editor, or created using a graphical modelling tool. The GARP model ingredients can be divided into three categories, which will be discussed in the following sections.

4.2.1 Aggregates

The first type of model ingredients are aggregates, the most complex constituents of a model, consisting of multiple model ingredients:

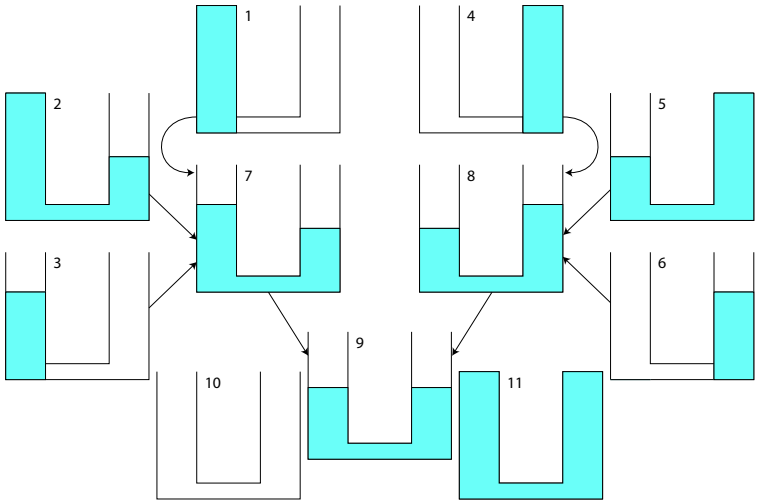

Figure 4: The full envisionment of a connected fluid container system.

Model Fragments are partial models composed of multiple ingredients. Model fragments have the form of a rule. This means that model ingredients are incorporated as either conditions or consequences. In table 1 the abilities of ingredients to be used as a condition or consequence are shown. As can be seen from the table, model fragments can be reused within other model fragments as conditions. Furthermore, subclasses of model fragments can be made, which augment the parent model fragment with new ingredients. Model fragments describe part of the structure and behaviour of a system in a general way. Actual system situations (see *scenarios*) matching this general situation description also have to contain the consequence ingredients.

There are three different kinds of model fragments. In the first, *static fragments*, all ingredients may occur except *agents* and *influences*. Static fragments are used to describe parts of the structure of the system. The second, *process fragments*, contain at least one influence, but no agents. These model fragments are used to describe processes which take place within the system. Finally, model fragments which contain an agent are called *agent fragments*, and may contain one or more influences. Agent fragments are used to describe the influences external entities (*agents*) have on the system.

Scenarios describe the actual state of a system, and can consist of all the ingredients which can be used as conditions in model fragments, except for other *model fragments* (see table 1). The use of ingredients in scenarios differs from their use in model fragments, as in scenarios ingredients are incorporated as facts instead of as conditions or consequences. Obviously, this allows scenarios to match on the conditions in model fragments. Scenarios are used as input for the qualitative simulator and function as the start states from which the rest of the behavioural graphs are generated.

4.2.2 Structural

The second category of modelling ingredients are the structural ingredients, which are used to describe the structure of the system:

Entities are the physical objects or abstract concepts that play a role within the system. These entities are arranged in an is-a hierarchy, which allows more general concepts higher in the hierarchy to be used in model fragments instead of more specific concepts below them. This allows for faster modelling, as there is no need to, for example, create a model fragment

describing flow for every kind of fluid. Instead, one model fragment describes flow and uses the generic concept fluid.

Configurations are used to model the relations between entities. For example, water being held by a container is represented by specifying a contains relation between them.

Attributes are properties of entities which do not gradually change during simulation. For this reason they do not have an associated *quantity space*. An attribute could be used to describe if a pan with boiling water is open or closed. Attributes are usually used as a condition in model fragments to indicate that a specific process is only active if the attribute has a certain value.

Agents are a special kind of entities. They are used to model entities outside of a natural system. They can have quantities influencing the rest of the system. A possible agent could be James constantly filling one of the water containers in the U-Tube system.

4.2.3 Assumptions

A model ingredient which describes neither the structural nor the behavioural aspects of a system are assumptions, as they constrain the behaviour of a model.

Assumptions are used to constrain the possible behaviour of a model. They can only be used as a condition, and are often combined with an inequality relation as a consequence. For example, a possible operating assumption might indicate that the containers in figure 4 are not empty. This would eliminate states {1,3,4,6,10}.

4.2.4 Behavioural

The final category of model ingredients are the behavioural dependencies. These relations are used to specify information about amounts, derivatives and the values of quantity spaces. The proportionalities and influences are often referred to as *causal relations* or *causal operators*.

Quantities are the changeable features of entities. For example, volume, height and pressure are quantities of a contained liquid. Each quantity has two associated quantity spaces: a definable one for the amount of the quantity, and the default quantity space {decreasing,zero,increasing} for the derivative.

Quantity Spaces specify a range of qualitative values a quantity amount or derivative can have. The qualitative values in a quantity space form a total order. Each qualitative value is either a point or an interval, and within the quantity spaces these two types consecutively alternate. An example of a quantity space for the height of contained liquid is {zero, positive, full}. Qualitative values having the same name do not necessarily have the same value. For example, two contained liquids with their height quantities equalling the value full do not necessarily have the same height. One exception is the value zero which specifies the turning point between positive and negative. That value is universally equal among quantity spaces.

Inequalitities ($<, \leq, =, \geq, >$) can be used in three different ways. Firstly, they are used to indicate relative differences in the sizes of amounts (relation between the quantities) or derivatives (relation between the derivatives) of quantities. Secondly, inequalities can be specified between the points of quantity spaces belonging to different quantities, to indicate, for example, that one qualitative value is bigger than the other. Finally, it is possible to specify that an amount or derivative has a value equal or inequal to a certain value. This relation between a quantity or derivative and a value could, for example, be used to indicate that a quantity amount has a value greater than zero.

Values are abbreviations for equality relations between a quantity and a qualitative value in its quantity space. Although an inequality relation between a quantity and an interval in its quantity space is impossible, it is possible to specify that a quantity has the interval as its value. The qualitative simulator converts these values to (possibly multiple) inequality relations. For example, a quantity set to an interval between two points using a value is mapped to the inequality relations $<$ the higher point, and $>$ to the lower point.

Influences, are directed relations between two quantities, and are either positive or negative. They are also referred to as direct influences. Influences are the cause of change within a model, and are therefore said to model a process. Depending on the amount of the source quantity and the type of influence the derivative of the target quantity either increases or decreases. An influence Flow I+ Amount causes the quantity Amount to increase if Flow is positive, and decrease if it is negative. For an influence I- this is just the opposite.

Proportionalities are also directed relations between two quantities, but do not cause quantities to change directly like influences. Instead, they propagate the effects of a process, i.e. they set the derivative of the target quantity depending on the derivative of the source quantity. For this reason, they are also referred to as indirect influences. Like influences, proportionalities are either positive or negative. A proportionality Amount P+ Height causes Height to increase if Amount increases, and decrease if Amount decreases. For a proportionality P- this is just the opposite. A proportionality is often visualised as $\propto$.

Correspondences are relations between qualitative values of quantity spaces belonging to different quantities, and can be either directed or undirected. The former means that when value α of quantity space X corresponds to value β of quantity space Y, you may derive that quantity space Y has value β, if quantity space X has value α. If the correspondence is undirected you may also derive the value α of quantity space X, if quantity space Y has value β. Correspondences also exist between two quantity spaces, indicating that each of the values of the quantity spaces of those quantities correspond to each other.

Plus/Min relations are used to calculate the sum or difference of two amounts or two derivatives. The plus/min relation can have an (inequality) relation itself. Usually an inequality relation is specified between the plus/min relation and a quantity to specify that the quantity is (in)equal to the result of the calculation. For example, the amount of flow in a U-tube system is equal to the difference between the two pressures of the liquids in the containers.

4.3 Qualitative Model Simulation

Using the knowledge representation presented in the previous section, a qualitative model can be developed. This model can be simulated using the qualitative reasoning engine GARP. The goal of QR simulation is to derive a state graph characterising the behaviour of the qualitative model from the model's structure.

The input of the simulator is one of the scenarios specified in the model, and becomes the begin situation in the state graph (see figure 5). The qualitative engine finds model fragments which describe the same situation as the scenario. The consequences of these model fragments are incorporated in the scenario. When the augmented scenario has been constructed, the proportionalities and influences are resolved. The resolved causal dependencies indicate how the quantities and its derivatives change, thus ending the state. This results in new states which are considered new scenarios. To generate the complete state graph the algorithm is repeated for each newly generated scenario.

Finding matching model fragments and incorporating the consequences can been seen as a classification task. Determining the successor states from a scenario is a prediction task. The

Possible conditions	**Impossible conditions**
Entities	Correspondences
Configurations	Proportionalities
Agents	Influences
Attributes	
Quantities	
Inequalities	
Min/Plus	
Assumptions	
Model Fragments	
Possible consequences	**Impossible consequences**
Entities (except in static MF)	Agents
Configurations (except in static MF)	Assumptions
Attributes	Model Fragments
Quantities	
Inequalities	
Min/Plus	
Correspondences	
Proportionalities	
Influences	

Table 1: Condition and consequence abilities of QR ingredients.

following sections discuss each of the inferences of the classification and prediction tasks in more detail.

4.3.1 The Classification Task

The classification algorithm takes a scenario or state as input and produces a complete description of the state, i.e. a state containing all consequences of model fragments applying to it and the inequalities derived by resolving causal operators. Making assumptions may result in multiple different descriptions of states.

Finding Candidate Model Fragments The first step in solving the classification task is finding candidate model fragments. Candidate model fragments are the model fragments which structurally match the scenario. The behavioural aspects of model fragments are ignored when searching for model fragments, as model fragments contain inequalities as conditions. These conditional inequalities might be true, but have to be derived from the other inequalities which apply to the scenario.

Testing Candidate Model Fragments The candidate model fragments which result from the previous step might or might not be consistent with the inequalities which are mentioned in their respective model fragments. This step tries to derive the conditional inequalities. If the inequalities can be deduced, they are incorporated in the scenario. If the inequalities cannot be derived, but are consistent with the other inequalities, they are stored as assumable inequalities. If the inequalities are inconsistent with the established inequalities, they are removed.

Testing Assumable Inequalities Searching for, and testing of, model fragments is repeated until no more new candidate model fragments can be found. These steps might have derived new inequalities, causing the assumable inequalities to become inconsistent. This step checks if the assumable inequalities are now derivable or inconsistent. Derivable inequalities are added to

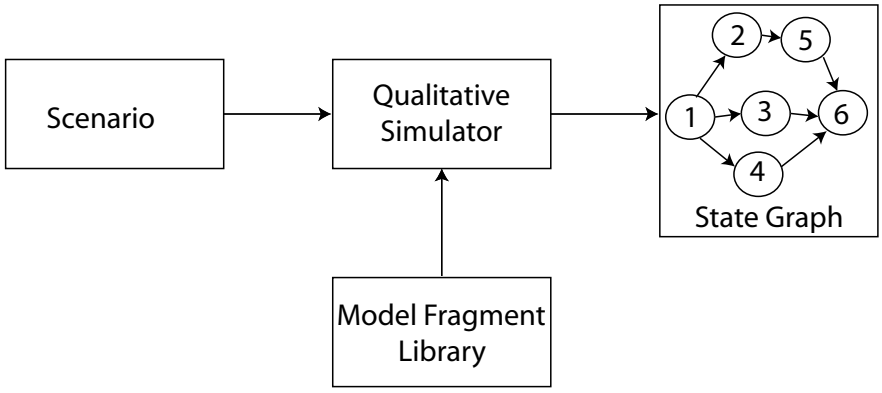

Figure 5: The qualitative simulator gets a scenario as input, and used the model fragment descriptions to generate a stategraph.

the scenario and removed from the assumable candidates, and inconsistent inequalities are just removed. The first iteration, no assumptions have been made (see next paragraph). If an assumed inequality is proven to be inconsistent, the complete scenario with that assumption is removed.

Make Assumptions Once this step is reached, there are no more candidate model fragments. Candidates are either (1) included in the scenario because their conditions are true, (2) ignored because their conditions are inconsistent with the state, or (3) in the set of assumable inequalities because the inequalities are consistent but not derivable from the scenario. This step finds consistent subsets of assumable inequalities. For each consistent subset of inequalities a copy of the scenario is made. The set of inequalities are merged with the copy of the scenario. The copied states go through the classification algorithm one last time to incorporate model fragments which match due to the assumed inequalities.

4.3.2 The Prediction Algorithm

The classification task has resulted in an augmented scenario (possibly multiple), which incorporate all the consequences of matching model fragments. The prediction algorithm takes this completed state description and identifies the successive states of behaviour and transitions to them.

Finding Terminations Termination rules are part of the qualitative engine. These rules indicate under what conditions states change. For example, if the value of a quantity is a point, and the derivative of that quantity is positive, the next state incorporates that quantity with the value on the interval above the point. Using this set of rules all the possible terminations of the scenario are gathered.

Ordering Transitions Not every termination in the set of possible terminations of a scenario applies. Some terminations occur simultaneously due to correspondences. Other have precedence over other terminations. For example, a transition from a point to an interval happens before the transition from a interval to a point. This step merges the terminations which have to occur simultaneously and afterwards removes the terminations which have no priority.

Generate successive states and transitions The final step generates the successive states and transitions using the final set of pruned and merged terminations.

4.4 Conclusions

To conclude, the questions posed for this chapter in the introduction are answered. *"What is the purpose of qualitative modelling?"* Qualitative modelling is a modelling paradigm which allows the derivation of the behaviour of system from a model. Unlike mathematical approaches, this paradigm requires no numerical data. This allows capturing of knowledge in domains in which numerical data is difficult to acquire.

"What does a QR model look like?" A qualitative model consists of model fragments which describe both the structural as the behavioural aspects of a system. The behavioural aspects consist of changeable features of entities and agents (quantities). Furthermore, the dependencies between these between these quantities are needed which indicate how they change.

"What are the kind of tasks the qualitative engine solves?" The qualitative reasoner solves a classification task, which matches scenarios on model fragments. The consequences are incorporated in the scenario. Afterwards the augmented scenario is used to solve a prediction task. The causal dependencies are resolved to determine the possible transitions to other states.

5 Ontology of Qualitative Model Ingredients

This chapter discusses the formalisation of the QR vocabulary and models in OWL. It presents both the generic ontology of QR model ingredients, which introduces the necessary vocabulary in terms of OWL, and the domain ontologies, which can be used to capture domain specific QR models (chapter 2). Furthermore, the implications of the ontology formalisations on using an OWL reasoner for QR problems is discussed.

5.1 Ontology Validation and Knowledge Representation Parser

In order to check if the ontologies defined in this chapter are correct three things were done. Firstly, and most importantly, the definitions of the QR concepts were discussed with QR researchers. This was done to make sure the ontology developer perspective is the same as the users of the ontology. Secondly, the consistency of the generic ontology was ensured by repeatingly testing it using an OWL reasoner[3]. Finally, a parser was developed which converts the knowledge representation of GARP to a knowledge representation in OWL. The domain ontologies which this parser creates refer to the vocabulary defined in the generic ontology. These models are used to check if the defined vocabulary is consistent. Using graphical OWL editors[4][5], an OWL reasoner, and some validators[6][7] the domain models (which include the generic ontology via the web) are analysed for their correctness.

5.2 A Hierarchy of Qualitative Model Ingredients

5.2.1 The Hierarchy

The formalisation of the QR domain starts with the ordering of the vocabulary in a class hierarchy. Figure 6 shows the taxonomy of the QR model ingredients. The taxonomy shows both the QR concepts and relations, as the latter are reified (treated as classes, see section 5.2.2). The top node of is called QualitativeModelIngredient, as every class is a possible ingredient of a qualitative model. The model ingredients are divided into the sets BuildingBlock and Aggregate. The former describes separate model ingredients, while the latter describes collections of related model ingredients. The aggregate concepts ModelFragment and Scenario are described in section 5.7. As mentioned in chapter 4, qualitative models describe both the structural and the behavioural aspects of systems. Therefore, the building blocks are subdivided into the sets Structural (section 5.4), Behavioural (section 5.6), and AssumptionType (section 5.5). Assumptions are considered to be separate, as they do not describe inherent aspects of the system.

One of the most difficult problems when developing an ontology is finding proper names for the defined concepts. Whenever a vocabulary concept is described in this thesis, its position in the hierarchy will be discussed along with the naming (if it differs from the standard QR vocabulary). The formalisation of the hierarchy in OWL, however, is straigh forward. The classes are defined by giving them an id, label, comment, and specifying their superclasses. As OWL assumes that the sets which the classes model overlap, the siblings on each level have been specified as being disjoint. The vocabulary hierarchy in OWL is shown in appendix B.1. In the rest of this chapter, the definitions of the concepts are extended (see section 3.7) by adding restrictions.

[3]Racer: http://www.sts.tu-harburg.de/ r.f.moeller/racer/
[4]Protege: http://protege.stanford.edu/
[5]Triple 20: http://www.swi-prolog.org/packages/Triple20/
[6]Wonderweb: http://phoebus.cs.man.ac.uk:9999/OWL/Validator
[7]BBN: http://owl.bbn.com/vowlidator/

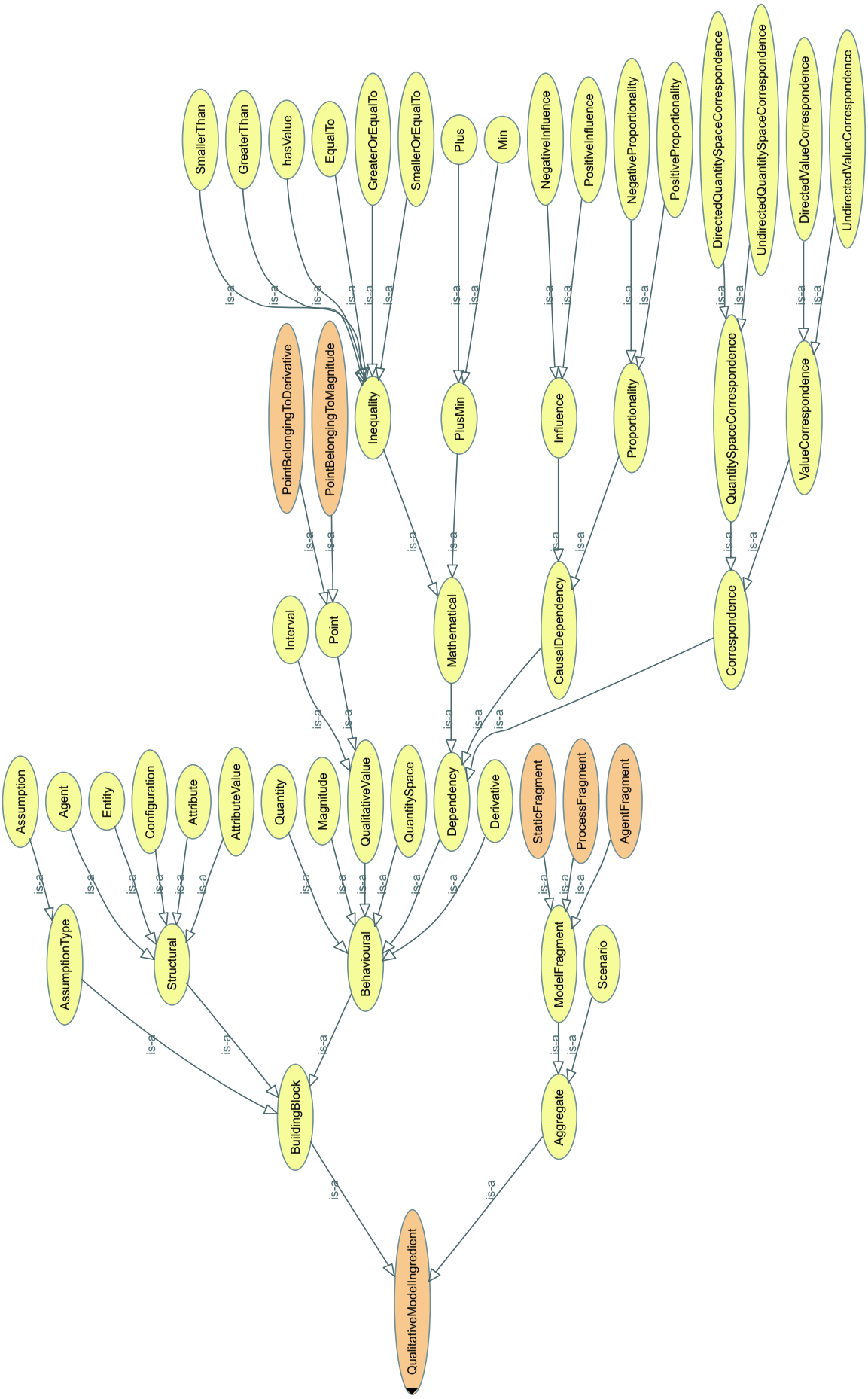

Figure 6: The QR ingredient taxonomy defining the QR vocabulary.

5.2.2 Reification of Relations

As mentioned in the previous section, all relations in the QR vocabulary have been reificated. This was necessary, as some QR relation ingredients are tertiary in nature. Furthermore, in order to render models in a graphical model editor, the position on the screen of each model ingredient must be stored. Treating the relations as classes (reification) allows instances of those relations to be connected to multiple objects, and have data type properties (for the position information). If model relations were not reificated, the ontology would become OWL Full, as properties cannot be treated as classes in OWL DL.

The reification of relations in OWL is a familiar problem and can be solved by using existing reification patterns [38]. There are two ways to reify a relation depending on the objects that take part in the relation. Generally, there is at least one independent object participating in a relation. An independent object exists without any other objects, while a dependent object, such as a property value, needs another object to exist. For the first pattern (figure 7) to apply, there must be only one independent object, which is the owner of the relation. If there is more than one independent object taking part in the relation, and there is no clear distinguishable owner, the second pattern is suggested to be used (figure 8).

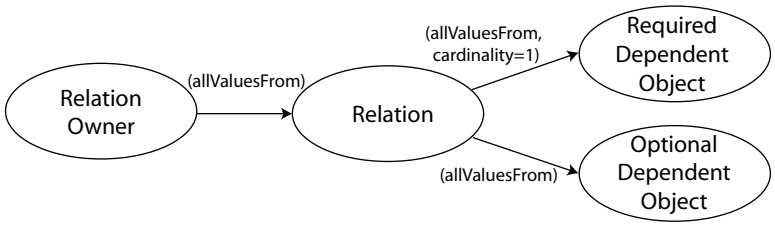

Figure 7: Relation Reification Pattern 1: One independent object which is the owner of the relation.

The single independent object in the first pattern (figure 7) must be the owner of the relation, as the dependent objects would not exist without it. Relations with only one independent object usually model properties of classes. Properties should have only one dependent object which is the value. For this reason, in this thesis relations from a reified relation to the dependent objects are defined as having cardinality 1. The standard pattern mentioned before uses functional properties instead of a cardinality restriction. The difference is that a cardinality restriction restricts the number of relations to 1, while a functional property indicates that all values of the property are the same individual. So, a cardinality imposes a restriction, while making the relation functional allows an inference. For the QR domain the restriction is more appropriate, as it decreases the number of possible models with the same meaning. Furthermore, it also possible for a reified relation to have an optional dependent object as a value. This restriction can be formalised by using maxCardinality, instead of cardinality=1. The formalisation of the pattern in OWL can be found in appendix A.1.

The second existing reification pattern applies when the relation has multiple independent objects as arguments, instead of just one. In this case, there is no clear owner of the relationship (see figure 8). For this reason, relations are modelled from the reified relation to the arguments. The inverse relations of the independent object should be defined, as the relations these objects take part in should be deducible. The formalisation of the pattern in OWL is shown in appendix A.2.

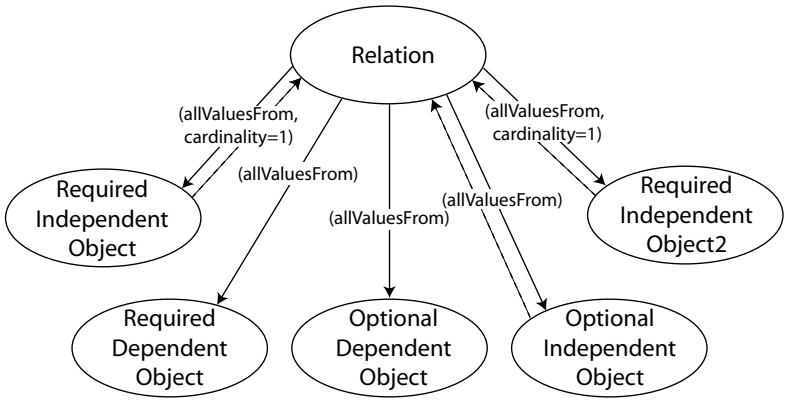

Figure 8: Relation Reification Pattern 2: Multiple independent objects with no clear owner. Note that the restrictions apply to the relations from the reificated relation.

OWL Element	Symbol	Example	Meaning
allValuesFrom	$\forall$	$\forall\ Person\ Human$	Every person is human
someValuesFrom	$\exists$	$\exists\ Person\ Male$	At least one person is male
hasValue	$\ni$	$isConsistent\ \ni\ true$	The consistent property must be true
cardinality	$=$	$hasWings\ =\ 2$	There must be exactly 2 wings
minCardinality	$\geq$	$hasEngines\ \geq\ 2$	There must be at least 2 engines
maxCardinality	$\leq$	$hasPassengers\ \leq\ 300$	There may be at most 300 passengers
complementOf	$\neg$	$\neg\ Male$	Everything not Male
intersectionOf	$\sqcap$	$Human\ \sqcap\ Male$	Every Human which is Male
unionOf	$\sqcup$	$Male\ \sqcup\ Female$	Anything that is either Male or Female
enumeration	$\{\ldots\}$	$\{John\ Mike\}$	The individuals John or Mike

Table 2: The equivalent OWL notation used in this section.

This second pattern is not applied in the QR ontologies as restriction semantics are not defined in the relation, but the class which owns the relation (see section 5.2.3). As this requires the ownership relations to point to the reificated relation, the first pattern is more appropriate.

5.2.3 Property Restrictions

The defined QR hierarchy specifies that classes inherit the properties defined in their superclasses. In this case the properties are the restrictions which specify how model ingredients may be related using object properties. This section describes how such restrictions have to be formalised. For brevity, the restrictions used in this chapter are formalised in the logical notation of the Protege OWL editor [27]. This notation is shown in table 2.

A common mistake made when modelling properties in OWL is using domain and range definitions of properties as restrictions [27]. These fields specify that the domain and range of the property *are* of a specific type. This means that if a relation playsInstrument with domain Person and range MusicalInstrument is used to model a robot playing piano, the statement is not inconsistent. It merely allows the reasoner to infer that the robot is a person. OWL users are advised not to specify these fields, as the resulting ontologies are very hard to debug [27].

The correct way to restrict the use of relations is by specifying the restrictions in the class

owning the relation. Examples of the specification of such restrictions have been shown in chapter 3. These restrictions become more complex when relations have been reificated, as it becomes possible to model the restrictions directly in the relation. This should not be done, as it makes it impossible to reuse the relation. If the playsInstrument relation from the previous paragraph would be reificated, and the domain and range restricted, the statement that the robot plays piano would become inconsistent.

Necessary playsInstrument Restrictions
$\forall\ hasPlayer\ Human$
$\forall\ hasInstrument\ Intrument$

In order to be able to reuse reificated patterns a new pattern was developed. Instead of modelling the restrictions in reificated relations, they should be defined in the classes using the relation. This is possible as OWL allows the creation of new anonymous class definitions within restrictions. In the robot pianist example, the person class would be restricted to having a relation with a reificated playsInstrument relation, which has a relation with an instrument. This definition allows reuse, as now the robot class can be formalised using the same restrictions. It is even possible to allow different instruments for the robot by replacing the Instrument class in the restriction. The OWL code corresponding to this example can be found in appendix A.3.

Necessary Human Restrictions
$\forall\ hasPlaysInstrumentRelation(playsInstrument\ \sqcap\ (\forall\ hasInstrument\ Instrument))$
Necessary Robot Restrictions
$\forall\ hasPlaysInstrumentRelation(playsInstrument\ \sqcap\ (\forall\ hasInstrument\ Instrument))$

5.3 Qualitative Model Ingredient

The qualitative model ingredients used in model fragments and scenarios have to be visualed. Therefore, the concept has the two data type properties has_xposition_on_screen and has_xposition_on_screen. As not every model ingredient has a specific position which needs to be stored (because the position is indicated by other ingredients), filling these values is not always necessary. Therefore, instead of cardinality restrictions, maxcardinality restrictions are used. The formalisation in OWL can been found in appendix B.2.

Necessary Qualitative Model Ingredient Restrictions:
$has\_xposition\_on\_screen\ \leq\ 1$
$has\_yposition\_on\_screen\ \leq\ 1$

5.4 Structural

This section describes structural ingredients set of the qualitative model ingredients, which are used to describe the structure of a system. This class encompasses the concepts (1) *entities*, (2) *agents*, (3) *configurations*, (4) *attributes* and (5) *attribute values*.

5.4.1 Entities, Agents, and their Relations

Entities describe the objects which exist in a system. Agents are very similar in the kind of relations they can take part in, but are used to model 'outside forces' on the system. They are both structural ingredients, and are alike in the kinds of relations they can take part in. It is possible for entities

and agents to have attributes (section 5.4.3) and quantities (section 5.6.1) as is shown in figure 9. These necessary restrictions are shown below, and formalised in OWL in appendix B.3. Entities and agents can have configuration relations with other entities and agents, which are described in section 5.4.2.

Necessary Entity and Agent Restrictions
$\forall$ *hasAttribute Attribute*
$\forall$ *hasQuantity Quantity*

Developers of qualitative models can define their own entities and agents in a subtype hierarchy. These entities and agent definitions are stored in the *domain ontology*. The formalisation of these hierarchies in OWL is similar to the formalisation of the QR vocabulary hierarchy (section 5.2.1). Again, the classes are defined with their respective superclasses, and disjointness axioms have to be added to indicate that individuals of classes cannot be members of the sibling classes. The top nodes of these hierarchies are the general *entity* and *agent* concepts defined in the *generic ontology*. Therefore, every model imports the generic ontology, and refers to these concepts though the correct namespace. An example of the formalisation of the entity and agent hierarchies, generated by the model parser, can be found in appendix C.1.

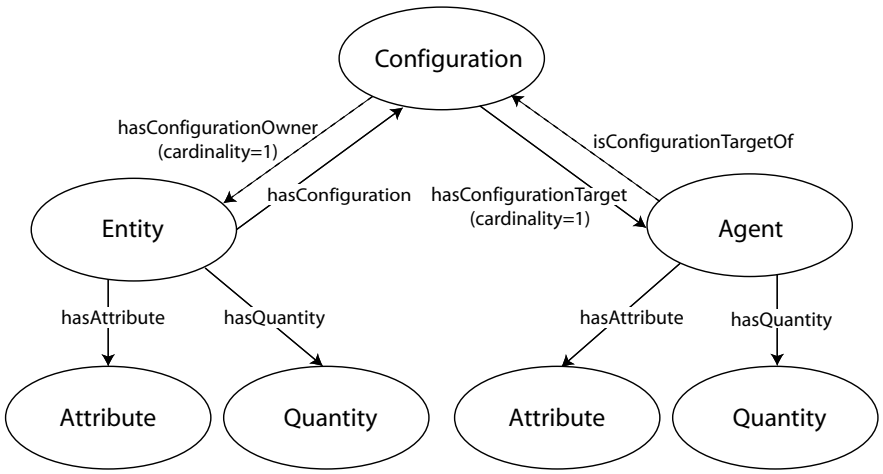

Figure 9: The possible relations of *entities* and *agents*. Note that both the entity may be changed to agent, and the agent into entity.

5.4.2 Configurations

Configurations are used to describe the structural relations between entities and agents. A configuration is a relation between two independent objects, but it is a directed relation. This means that the owner of the relation can be clearly identified. This means that both the first (a clear owner of the relation can be identified) and the second relation reification pattern (participants in the relation are independent) could apply. As mentioned in section 5.2.2, the second reification patterns is not used. Therefore the configuration is modelled as a mixture of the two patterns, as is shown in figure 9. The difference with the first pattern is that the inverse relations have been defined, so the configurations which entities and agents take part in can be derived.

Configurations have exactly one owner and one target, therefore the hasConfigurationTarget and hasConfigurationOwner relations are defined as having cardinality 1. By defining inverse relations

the owner belonging to the configuration, and the configuration which targets an entity or agent can be derived. The OWL definitions can found in appendix B.4.

Configuration Restrictions
$hasConfigurationOwner \ = \ 1$
$hasConfigurationTarget \ = \ 1$

As mentioned in section 5.2.3, the restrictions which apply to relations are defined in the class which utilises them. In this case the semantics of the configuration relation are defined in the agent and entity classes. This allows new model ingredients to be defined which also use the configuration relation. Entities and agents can participate in configuration relations with other entities and agents. The formalisation can be found in appendix B.4.1.

Necessary Entity and Agent Restrictions
$\forall \ hasConfiguration \ (Configuration \ \sqcap$
$(\forall \ hasConfigurationTarget \ (Entity \ \sqcup \ Agent)))$

Developers of qualitative models can define their own configurations. In contrast with entities and agent, configurations are not arranged in a taxonomy. As a result every configuration is a subclass of the concept *configuration* defined in the generic ontology. An example of some configuration definitions in OWL can be found in appendix C.2

5.4.3 Attributes

Attributes describe the features of entities and agents that do not change gradually. Models can define their own attributes, which consist of an attribute name, and its possible values. Attributes are considered to be properties of entities and agents, and should have exactly one attribute value. As OWL does not make a distinction between relations between objects and properties of objects[8], a pattern has to be used to formalise properties in OWL. An existing pattern to model property values uses an enumeration of individuals [40].

Property Values as an enumeration of individuals In the enumeration pattern the values of a property are considered to be a set of individuals, as shown in figure 10. As a result the values are unique in the ontology, and objects with the property all refer to a value from the same set of individuals. It is necessary to explicitly state that the values are different using the owl:differentFrom statement, as OWL does not make the Unique Name Assumption (two individuals with different names are not necessarily different objects). The formalisation of this pattern in OWL can be found in appendix A.4.1.

Back to the formalisation of attributes. The attribute value is defined to be an enumeration of individuals (see figure 11). This is possible as attribute values do not have a position on the screen which has to be stored. This allows the same value set to be used for each instance of an attribute. Attribute relations are a typical example of reification pattern 1. The attribute itself is an independent object, while the attribute values are dependent on the existence of the attribute. Since an attribute always has a relation with an attribute value, the semantics are stored in the attribute class instead of in the classes having attributes. The formalisation of the attribute and attribute value concepts in OWL is shown in appendix B.5.

[8]Not to be confused with ObjectProperties in OWL

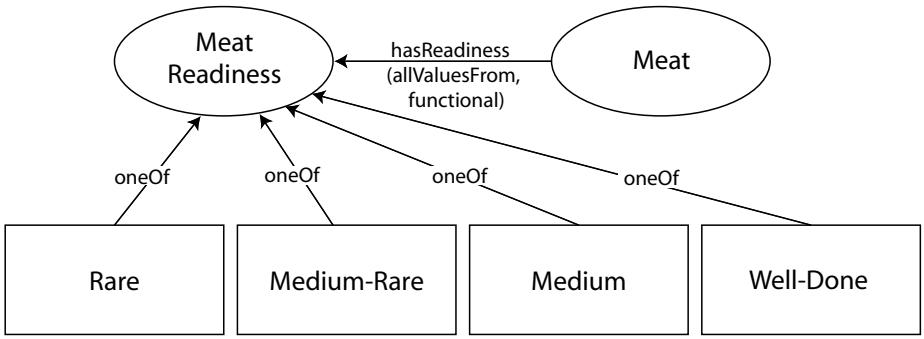

Figure 10: Values as an enumeration of individuals.

Attribute Restrictions
$\forall\ hasAttributeValue\ AttributeValue$
$\forall\ hasAttributeValue\ =\ 1$

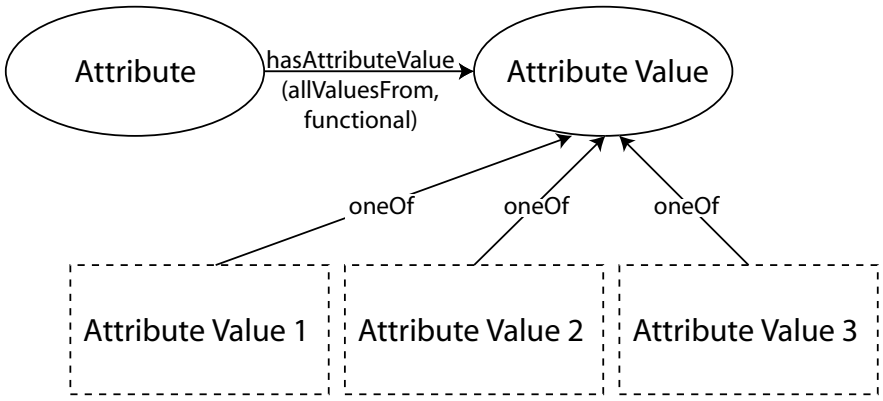

Figure 11: The attribute components and its relations.

The attributes defined in the domain ontology have some further restrictions. The attribute is prohibited to have any other attribute value, other than an instance of the associated attribute value class. The attribute value class is defined as the enumeration of the possible values. Examples of attribute and attribute value definitions in a model context can be found in appendix C.4.

OpenOrClosed Restrictions
$\forall\ hasAttributeValue\ OpenOrClosedValue$
OpenOrClosedValue Restrictions
$\{Open, Closed\}$

5.5 Assumption Types

As it is likely that new types of assumptions will be introduced in the QR vocabulary, the AssumptionType concept is defined. For now, the only class which is part of this set is the generic assumption. Assumptions are neither structural nor behavioural ingredients, as they describe knowledge about the model, and not about the system.

Assumptions are used as conditions in model fragments to assert that something is true. They are usually used in combination with inequalities to reduce the possible behaviour of a system. As assumptions have no further relations with other QR ingredients, no restrictions have to be imposed in the ontology. Model builders can define their own assumptions in the same way they can define entities and agents. The formalisation of this subsumption hierarchy in OWL is equivalent to those of the entities and agents, as can be seen in appendix C.3.

5.6 Behavioural

Behavioural ingredients describe the model ingredients used to describe the behaviour of a system. They include: (1) *Magnitudes*, (2) *derivatives*, (3) *quantities*, (4) *quantity spaces*, (5) *qualitative values*, and (6) *dependencies*. Qualitative values can be further divided into *points* and *intervals*.

Dependencies are the possible relations between the behavioural ingredients of a system. They are used to model the processes causing change, constrain what changes can occur, and how these changes occur. The dependency ingredients include: (1) the *causal dependencies: proportionality* and *influence*, (2) *correspondences*, and (3) the *mathematical dependencies: plus/min* and all *(in)equalities*.

5.6.1 Quantities, Magnitudes, Derivatives and Quantity Spaces

Quantities describe the changeable features of entities and agents. These behavioural ingredients consist of exactly one magnitude and exactly one derivative (see figure 12). Both the magnitude and the derivative have exactly one quantity space. The formalisation of the quantity, magnitude and derivative restrictions in OWL is shown in appendix B.6.

The magnitude is actually the zero derivative of the quantity, while the derivative is the first derivative. Forbus uses the term *amount* to refer to the zero derivative, and magnitude to refer to the value of the magnitude or derivative [13]. The term amount is not used in our approach, because it is ambiguous. Next to Forbus' use, it could also be used to refer to the quantity "amount". Another alternative is the term value, but that word has the problem that the qualitative values within a quantity space are also referred to as values. Magnitude does not have these problems, and is an appropriate name in this context. Note that the qualitative simulator GARP does not distinguish a separate magnitude concept from a quantity. In that context, the relations meant for the magnitude are established on the quantity. Quantities can have proportionality and influence relations (section 5.6.3). Magnitudes and Derivatives can participate in (in)equality relations (section 5.6.5) and Plus/Min relations (section 5.6.6).

Necessary Quantity Restrictions:
$\forall\ hasMagnitude\ Magnitude$
$hasMagnitude\ =\ 1$
$\forall\ hasDerivative\ Derivative$
$hasDerivate\ =\ 1$

Necessary Magnitude Restrictions:
$\forall\ hasQuantitySpace\ QuantitySpace$
$hasQuantitySpace\ =\ 1$

Necessary Derivative Restrictions:
$\forall\ hasQuantitySpace\ QuantitySpace$
$hasQuantitySpace\ =\ 1$

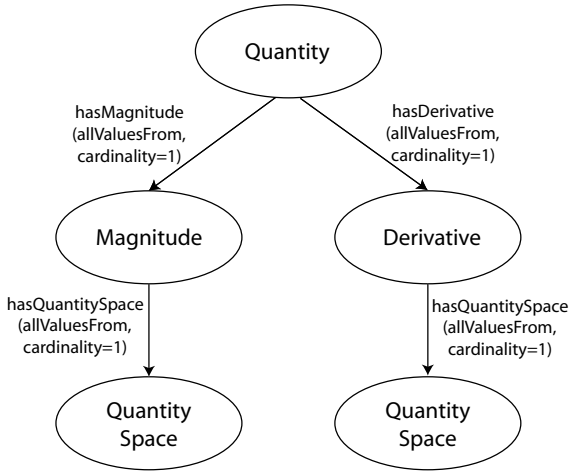

Figure 12: The quantity has one magnitude and one derivative, which both have quantity spaces.

Qualitative model builders can define their own quantities. A quantity can have a number of allowed quantity spaces (section 5.6.2). This information has to be stored in the specific quantity concept in the domain ontology. The formalisation in OWL is done using a restriction. The quantity has a magnitude, which has a quantity space relation with one of the quantity spaces in a union. The formalisation of a quantity space in OWL is shown in appendix B.6.

Necessary Flow Restrictions:
$\exists\ hasMagnitude(Magnitude\ \sqcap$
$(\forall\ hasQuantitySpace(Minimumnegativezeropositivemaximum\ \sqcup$
$Negativezeropositive\ \sqcup\ Zeropositivemaximum)))$

5.6.2 Quantity Spaces and Qualitative Values

The quantity space is a behavioural ingredient which defines the possible qualitative values a quantity can have. A quantity space consists of at least one qualitative value. These values are also behavioural ingredients, and can be either a point or an interval. The quantity space describes a total order, meaning that a magnitude or derivative can only change to a value directly above or below its current value. Quantity spaces can participate in a correspondence relations (section 5.6.4).

Necessary QuantitySpace Restrictions
$\forall\ hasQualitativeValue\ QualitativeValue$
$hasQualitativeValue\ \geq\ 1$

As each qualitative value in a model fragment can participate in (in)equality relations, it is impossible to formalise them as an enumeration of individuals. It is more graceful to model each value which can participate in a relation as a separate individual. Therefore, "the property values as a set of individuals" pattern cannot be used. Another existing pattern which is more appropriate formalises each value as a class [40].

Property Values as classes Values of properties can be thought of as a set of subclasses forming a parent class (see figure 13). This class binds all the possible value classes of the property using the owl:unionOf construct. It is necessary to explicitly state that the subclasses are disjoint, as

otherwise a property could have two values at the same time (because they are the same, i.e. the value is an instance of both subclasses). In contrast with the 'property values as individuals' pattern, values are not unique, but a new instance of the value is created for each property instance. The OWL code corresponding to this pattern can be found in appendix A.4.2. A problem with this pattern is that the property values do not have an explicit order, which is needed for quantity spaces. An existing pattern which might solve this problem is to model the values as a sequence in a list.

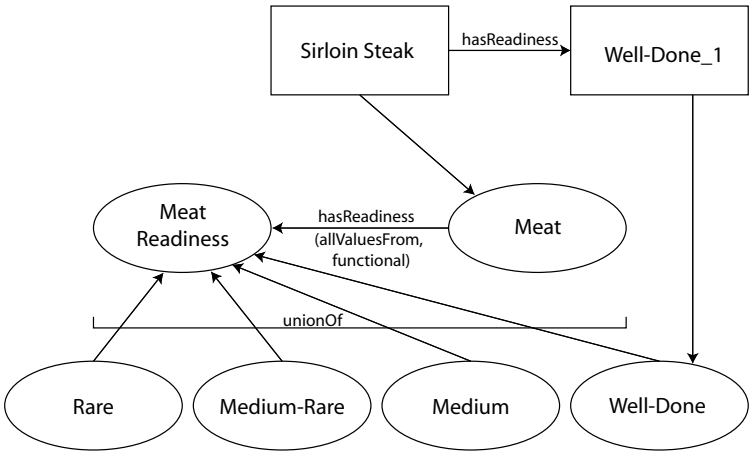

Figure 13: Property values as the instances of classes which form a union.

Property Values as a Sequence in a List This pattern is described in the n-ary relations document of the Semantic Web Working Group [38]. If the possible values of a property have a strict sequence, the previous patterns are not expressive enough, as they do not enforce an ordering. A possible solution to this problem is to model the values as a list. Unfortunately, using rdf:List in an OWL ontology causes it to become OWL Full. A solution is to model a list in OWL (see figure 14), which is then used to store values.

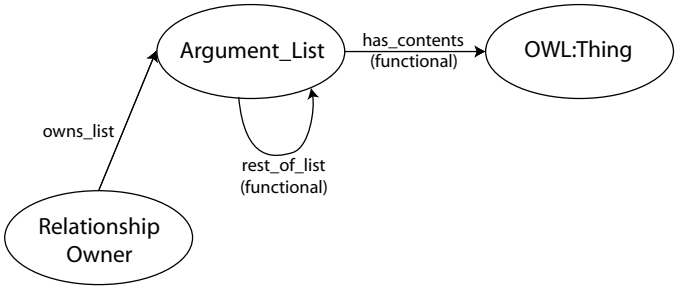

Figure 14: The definition of a list in OWL.

A list consists of a number of arguments, each pointing to the next item in the list. Each argument item has as has_content relation with the an object. The list pattern can be used in

combination with both the "property values as an enumeration of individuals", and the "property values as a set of classes" patterns. To formalise a quantity space the latter pattern must be used. The values are ordered in a list to model their sequence, as is shown in figure 15. The OWL formalisation of this pattern can be found in appendix A.4.3.

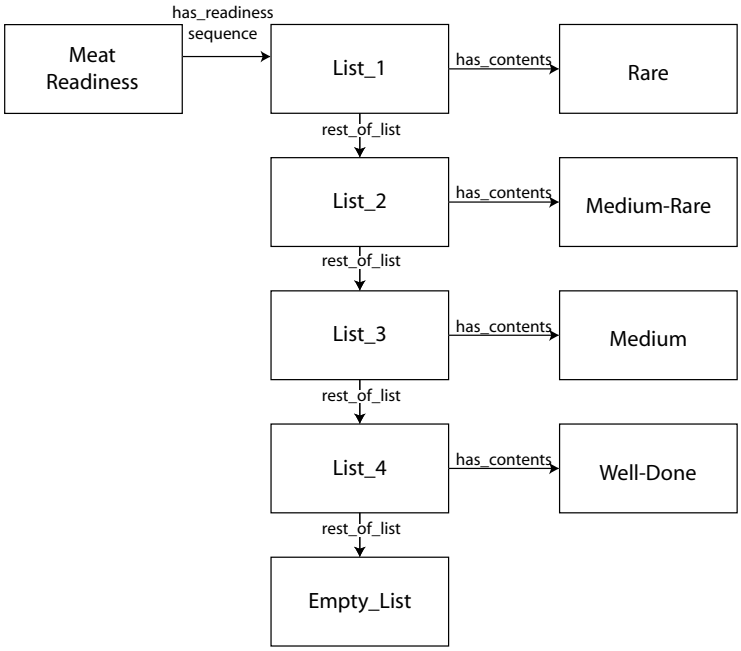

Figure 15: The application of a list in OWL.

However, there are two problems with the list pattern. Firstly, a list has no ontological meaning, as it is a data structure. A list with the cities Amsterdam, Brussel, and Paris has little meaning. They could indicate a travel route, cities with the around the same amount of inhabitants, or something else entirely. Secondly, it is awkward to determine the owner of a list from one of the possible values. One has to "reason through" all the values. In order to solve these problems a new pattern had to be developed; the ontological sequence.

Ontological Sequence In order to formalise a quantity space the ordering of the values has to be made explicit. The quantity space is connected with its points and intervals using containsQualitativeValue relations, as is shown in figure 16. The ordering is established by using inequalities (section 5.6.5). Each consecutive value in the order must have another type than the previous one. For that reason, the (in)equality restrictions are formalised in the intervals. This still allows connecting two points, but that is necessary for the other (in)equality relations (section 5.6.5). The point Zero is universal among quantity spaces, and is therefore defined in the generic ontology. The formalisation of the quantity restrictions can be found in appendix B.7. Qualitative model builders can specify their own quantity spaces. These have the form shown in figure 16. An OWL formalisation of an example quantity space can be found in appendix C.5.

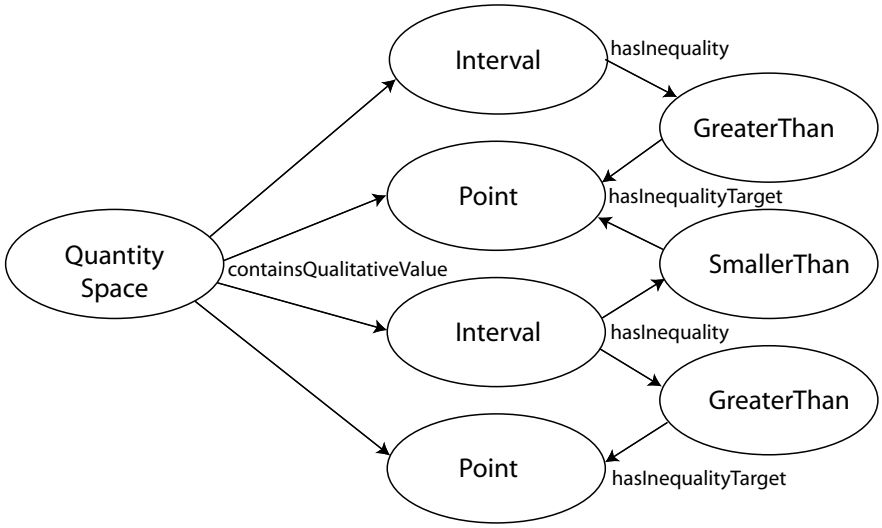

Figure 16: The formalisation of a quantity space and its values using inequalities.

Necessary Quantity Restrictions
$\forall\ containsQualitativeValue\ QualitativeValue$
$containsQualitativeValue\ \geq\ 1$

Necessary NegativeZeroPositive Restrictions
$\exists\ containsQualitativeValue(Positive \sqcap$
$\quad (\exists\ hasInequality(GreaterThan\ \sqcap\ (hasInequalityTarget\ \ni\ Zero))))$
$\exists\ containsQualitativeValue(Negative\ \sqcap$
$\quad (\exists\ hasInequality(SmallerThan\ \sqcap\ (hasInequalityTarget\ \ni\ Zero))))$
$containsQualitativeValue\ \ni\ Zero$

5.6.3 Proportionalities and Influences

Proportionalities and Influences are relations between quantities which indicate what quantities in a model change. These relations can be either positive or negative. A positive proportionality indicates that the derivative of the target quantity is positive if the derivative of the origin quantity is positive, and is negative if the derivative of the origin quantity is negative. For a negative proportionality this is just vice versa. A positive influence indicates that the target quantity derivative is positive if the magnitude of the origin quantity is greater than zero, and negative if it less than zero. For the negative influence this is just the other way around.

The semantics of the causal dependencies are modelled in the owner class, in this case the quantity class. Quantities can have influences and proportionalities as causal dependency relations, and they must have exactly one quantity as a target (see figure 17). The formalisation in OWL is shown in appendix B.8.1.

Necessary Quantity Restrictions

$\forall\ hasCausalDependency($
$(Influence\ \sqcup\ Proportionality) \sqcap$
$(\forall\ hasCausalDependencyTarget\ Quantity)\ \sqcap$
$(hasCausalDependencyTarget\ =\ 1))$

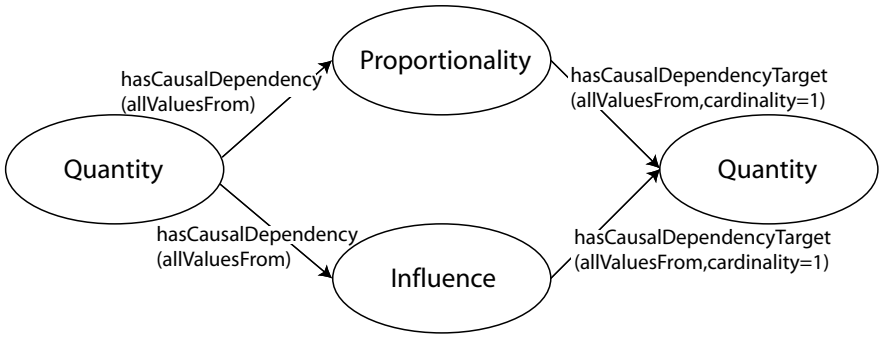

Figure 17: The use of proportionality and influences relations. Note that the restrictions applied to the hasCausalDependencyTarget relation are modelled in the Quantity class which owns the relation, and not in the influence or proportionality class.

5.6.4 Correspondences

Correspondences specify that values occur simultaneously. They normally occur between qualitative values of different quantity spaces, but it is also possible to create a correspondence relation between quantity spaces themselves. Those relations are an abbreviation for value correspondence relations between each of the quantity values of the different quantity spaces. Directed and undirected correspondences are distinguished. The former states that the target value may be derived from the origin value, while the latter allows the derivation of the value in both directions.

The semantics of quantity space correspondences is unclear when the quantity spaces have a different amount of values, as it is unknown between which values value-correspondences would exist. The only way to formalise this in OWL is to create classes for each quantity space size, and restrict correspondences to members with the same number of qualitative values. The correct formalisation would take an infinite number of classes, therefore this restriction is not implemented. It would be beneficial if it would be possible to express in OWL that a domain and range should have the same cardinality. Another problem is that correspondences, just as configurations, should not be reflexive. It is a known problem that this restriction is inexpressible in OWL. For value correspondences a restriction is needed that restricts value correspondences to members of different quantity spaces. As OWL restrictions can only state that a range has to be of a specific type, this is impossible to implement. This leaves the basic restrictions that correspondences should be between members of the same class, and should have exactly one target, as is shown in figure 18. The formalisation in OWL is shown in appendix B.9.

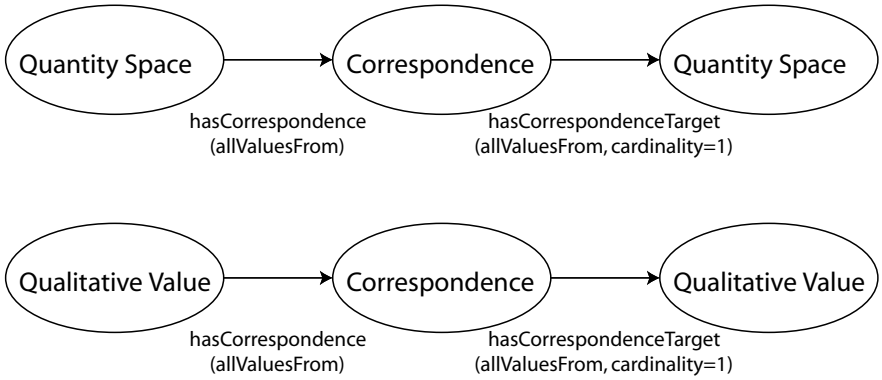

Figure 18: The only valid correspondence relations are between two quantity spaces, or two qualitative values.

QualitativeValue Restrictions:

$$\forall\ hasCorrespondence(ValueCorrespondence \sqcap (\forall\ hasCorrespondenceTarget\ QualitativeValue)\ \sqcap (hasCausalCorrespondenceTarget\ =\ 1))$$

Quantity Space Restrictions:

$$\forall\ hasCorrespondence(QuantitySpaceCorrespondence \sqcap (\forall\ hasCorrespondenceTarget\ QuantitySpace)\ \sqcap (hasCausalCorrespondenceTarget\ =\ 1))$$

5.6.5 Inequalities

Inequalities are dependencies which can be used by a lot of ingredients. They indicate the difference between, or equality of values. The same reificated (in)equality relations SmallerThan, SmallerOrEqualTo,EqualTo,GreaterOrEqualTo and GreaterThan are reused for each possible domain/range combination. This reuse is possible as the restrictions are formalised in the classes owning the relations, as was discussed in section 5.2.3.

The first (in)equality relation which is described are the inequalities between points. These relations are only valid if both points belong to either a magnitude or a derivative. As it is undesirable to have to distinguish between these type of points in a qualitative model, a pattern is developed to formalise these specific restrictions.

Restrictions for Classes Meeting Specific Conditions There are cases when the relations of an individual have to be restricted depending on the relations that individual has. For example, a workspace meant for a desktop computer should also contain a monitor, but a workspace meant for a laptop does not. Because of the option between a laptop or computer necessity of having a monitor cannot be modelled as a condition in the workspace class.

This problem can be solved by defining a new class workspaceWithDesktop in which necessary and sufficient, and necessary conditions are combined. Recall that necessary conditions are restrictions a consistent individual has to adhere to. Being an individual (or subclass) of such a class implies that its conditions apply ($class \implies conditions$). Necessary and sufficient conditions should be read as an equivalence relation. Fulfilling the conditions means the individual is an instance of the class, and being an individual of the class means the conditions apply ($class \iff conditions$).

The class WorkspaceWithDesktop (see figure 19) has "being of the class Workspace" and "having a desktop as a computer" as necessary and sufficient conditions, meaning that individuals fulfilling these conditions are classified as being a WorkspaceWithDesktop. Individuals of this class have to fulfil the necessary conditions, in this case containing a monitor. As shown combining necessary and sufficient, and necessary conditions allows modelling additional restrictions to individuals adhering to certain conditional relations.

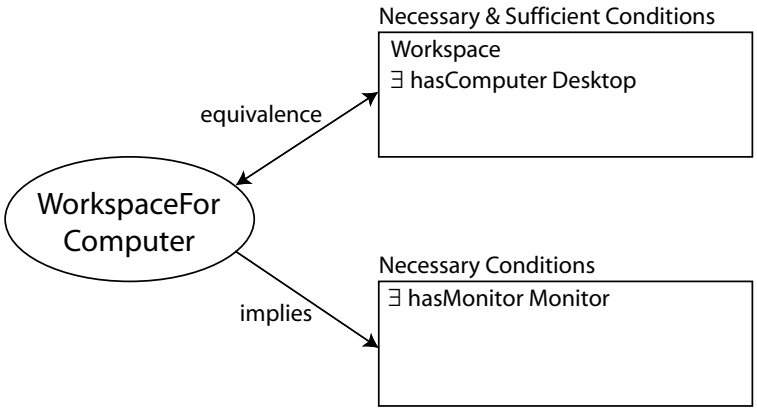

Figure 19: Combining necessary and sufficient, and necessary conditions.

Inequalities from Points Belonging to Magnitudes or Derivatives Points belonging to magnitudes (derivatives) can be classified as such by combining necessary, and necessary and sufficient conditions. Points belonging to magnitudes (derivatives) can have (in)equality relations with other points belonging to magnitudes (derivatives). Using the presented pattern necessary and sufficient conditions are stated to classify points, after which the necessary restrictions on the (in)equality apply. The range of the (in)equality should not be a value in the same quantity space, but this is, as mentioned before, not expressible in OWL. The restrictions can be seen in figures 20 and 21. The formalisation in OWL is shown in appendix B.10.1.

PointBelongingToMagnitude Necessary and Sufficient Conditions:
$\exists\ belongsToQuantitySpace\ (QuantitySpace\ \sqcap\ (\exists\ isQuantitySpaceOf\ Magnitude))$
PointBeloningToMagnitude Necessary Conditions:
$\forall\ hasInequality\ PointBelongingToMagnitude$

PointBelongingToDerivative Necessary and Sufficient Conditions:
$\exists\ belongsToQuantitySpace\ (QuantitySpace\ \sqcap\ (\exists\ isQuantitySpaceOf\ Derivative))$
PointBeloningToDerivative Necessary Conditions:
$\forall\ hasInequality\ PointBelongingToDerivative$

Inequalities originating from Magnitudes and Derivatives Magnitudes (derivatives) can have (in)equality relations with other magnitudes (derivatives) (figure 22 and 23). Again non-reflexivity is impossible to formalise. Magnitudes (derivatives) can also have (in)equality relations with qualitative values from their own quantity space (figure 24 and 25). As it is not possible to refer to the individual to which the restriction applies, this restriction is also not formalised. As a result,

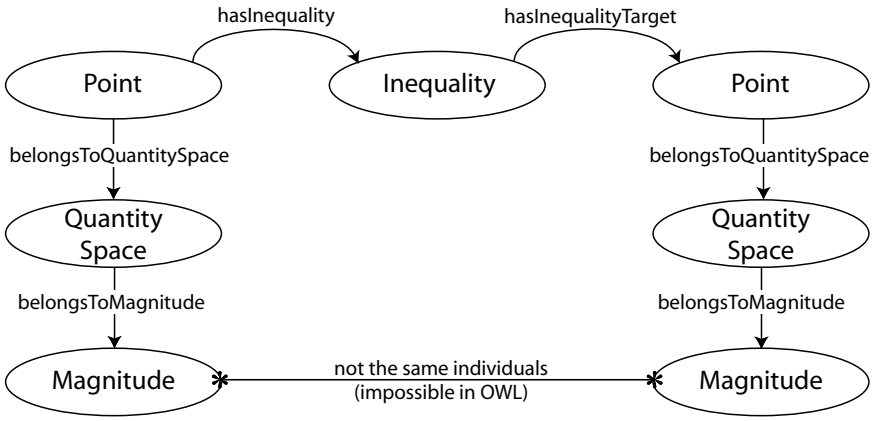

Figure 20: Points belonging to magnitudes can only have (in)equality relations with points belonging to magnitudes.

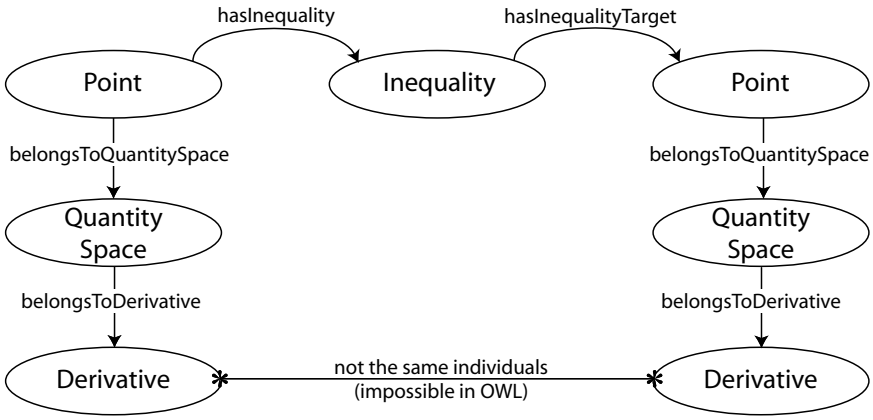

Figure 21: Points belonging to derivatives can only have (in)equality relations with points belonging to derivatives.

undesirable (in)equality relations can be formalised, while the ontology remains consistent. Firstly, reflexive inequalities can be described, which are not valid in GARP. Secondly, inequalities from magnitudes (derivatives) to qualitative values of other magnitudes (derivatives) can be described, which are also not allowed in GARP. The formalisation in OWL is shown in appendix B.10.2.

Magnitude Necessary Conditions:

$\forall\ hasInequality(Inequality\ \sqcap\ (\forall\ hasInequalityTarget\ (Magnitude\ \sqcup\ Point)))$

Derivative Necessary Conditions:

$\forall\ hasInequality(Inequality\ \sqcap\ (\forall\ hasInequalityTarget\ (Derivative\ \sqcup\ Point)))$

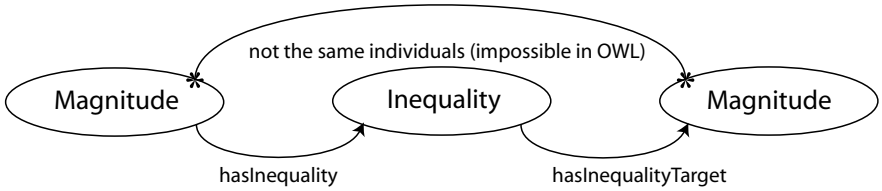

Figure 22: A valid (in)equality relations between two magnitudes.

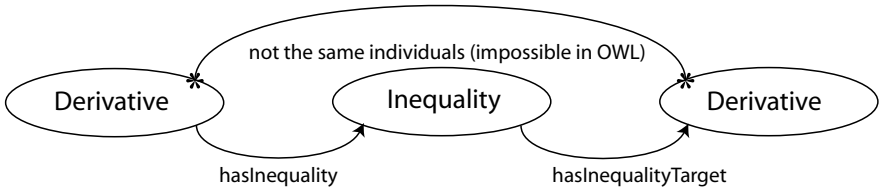

Figure 23: A valid (in)equality between two derivatives.

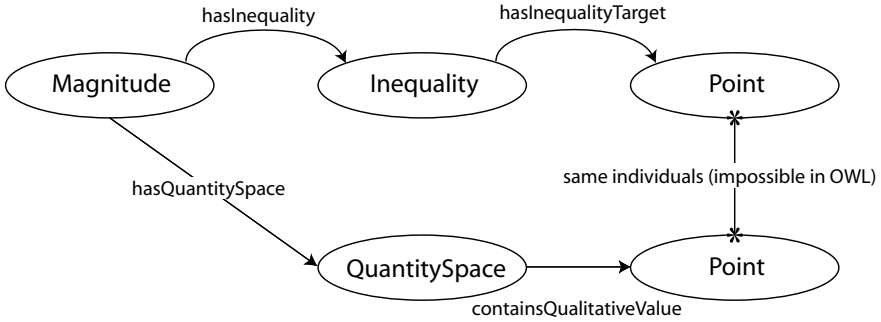

Figure 24: An (in)equality relations between a magnitude and a point.

Inequalities Used to Model the Total Order in Quantity Spaces The inequalities used to model the total order in quantity spaces were already discussed in section 5.6.2. Given that inequalities between points are valid, it is impossible to enforce the strict alternation between points and values in a quantity space. In order not to complicate the restrictions in points, the (in)equality relations used to specify the quantity space order are modelled in the interval class (see figure 16). The formalisation can be found in appendix B.10.3.

Interval Necessary Conditions:
$\forall\ hasInequality(Inequality\ \sqcap\ (hasInequalityTarget\ \geq\ 1)\ \sqcap$
$(\forall\ hasInequalityTarget\ Point))$

The hasValue Relation In order to facilitate easier modelling of (in)equality relations, GARP introduces hasValue relations. These relations are visualised as arrows which point at values of quantity spaces, indicating that the magnitude or derivative has that specific value. Unlike inequalities, hasValue relations may point to intervals. The qualitative reasoner translates hasValue relations with intervals to 2 inequalities. One stating that the magnitude (or derivative) has a value smaller than the point above the interval, and one stating the value is greater than the point below the interval. The formalisation is shown in appendix B.10.4.

Necessary hasValue relations:
$\forall\ hasValueTarget\ QualitativeValue$

5.6.6 PlusMin Relations

Plus and min relations are tertiary relations which add or substract two values, and indicate that a magnitude, derivative or point is equal to the result. Two different plus/min relations may be distinghuised. In the first, only magnitudes and point belonging to magnitudes may participate in the relation (see figure 26). In the second only derivatives may participate in the relation (see figure 27). Both plus/min relations should have exactly one left-handside and one right-handside.

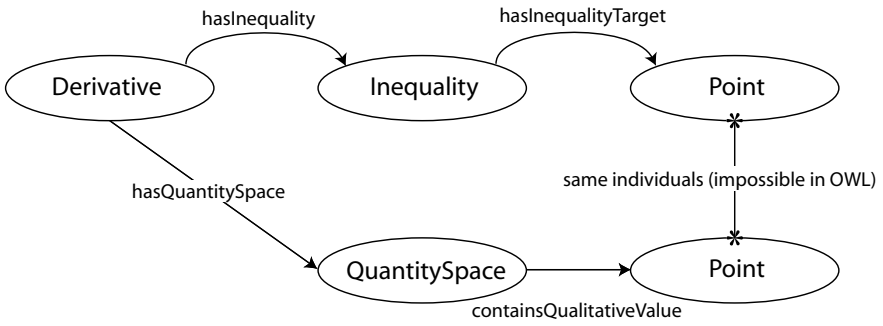

Figure 25: An (in)equality relations between a derivative and a point.

Any number of inequalities may be placed from the plus/min relations to valid targets. The restrictions for the first plus/min relation are formalised in both the PointBelongingToMagnitude class and the Magnitude class. The restrictions for the second is formalised in the derivative class. The OWL formalisation of plus/min relations can be found in appendix B.11.

PointBelongingToMagnitude Restrictions:

$$\forall\ isLefthandSideOf(PlusMin\ \sqcap\ (hasLefthandSide\ =\ 1)\ \sqcap\ (\forall\ hasRighthandSide\ (Magnitude\ \sqcup\ PointBelongingToMagnitude))\ \sqcap\ (hasRighthandSide\ =\ 1)\ \sqcap\ (\forall\ hasInequality(Inequality\ \sqcap\ (\forall\ hasInequalityTarget(Magnitude\ \sqcup\ PointBelongingToMagnitude)))))$$

Magnitude Restrictions:

$$\forall\ isLefthandSideOf(PlusMin\ \sqcap\ (hasLefthandSide\ =\ 1)\sqcap\ (\forall\ hasRighthandSide(Magnitude\ \sqcup\ PointBelongingToMagnitude))\sqcap\ (hasRighthandSide\ =\ 1)\ \sqcap\ (\forall\ hasInequality(Inequality\ \sqcap\ (\forall\ hasInequalityTarget(Magnitude\ \sqcup\ PointBelongingToMagnitude)))))$$

Derivative Restrictions:

$$\forall\ isLefthandSideOf(PlusMin\ \sqcap\ (hasLefthandSide = 1)\ \sqcap\ (\forall\ hasRighthandSide\ Derivative)\ \sqcap\ (hasRighthandSide\ =\ 1)\ \sqcap\ (\forall\ hasInequality(Inequality\ \sqcap\ (\forall\ hasInequalityTarget\ Derivative))))$$

5.7 Aggregate

Aggregates are model constituents consisting of multiple QR ingredients. *Model Fragments* and *Scenarios* are aggregate types. Model Fragments can be further subdivided into (1) *static fragments*, (2) *process fragments*, and (3) *agent fragments*.

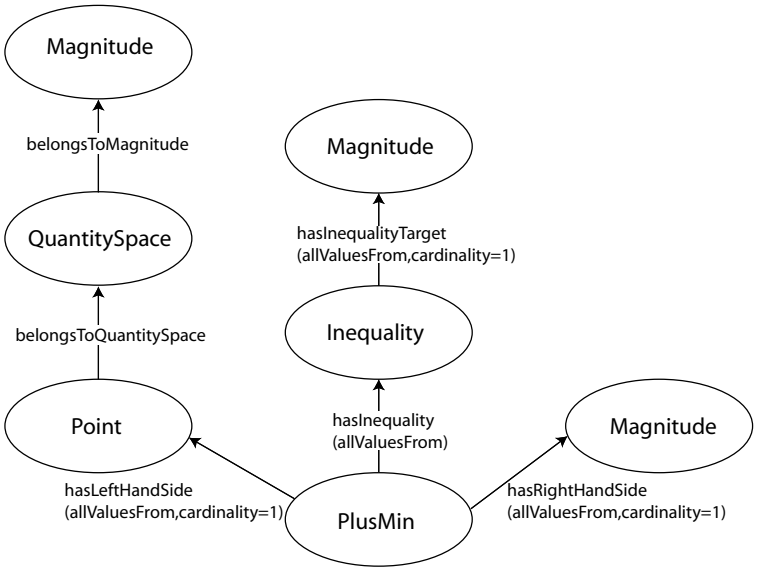

Figure 26: The possible Plus and Min relations between magnitudes and points belonging to magnitudes. Note that every magnitude and the point belonging to the magnitude could be interchanged.

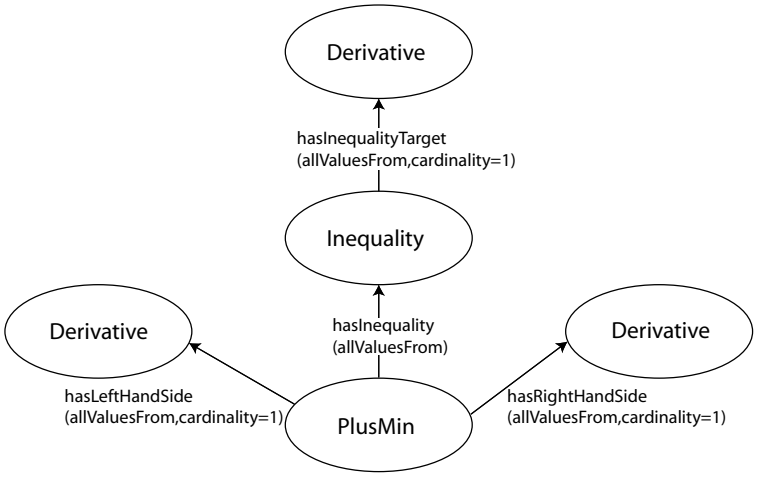

Figure 27: The possible Plus and Min relations between derivatives.

5.7.1 Model Fragments

Model fragments consist of multiple model ingredients. As these ingredients are incorporated as either conditions or consequences, model fragments have the possible relations hasCondition and hasConsequence. As was already discussed in section 4.2 there are restrictions on what kind of ingredients may be used as conditions and consequences in model fragments. These restrictions were described in table 1. The formalisation of these restrictions in OWL can be found in appendix B.12.1.

Necessary Model Fragment Restrictions

$\forall\ hasCondition(Structural \sqcup Behavioural \sqcup$
$AssumptionType \sqcup Inequality \sqcup PlusMin \sqcup ModelFragment)$
$\forall\ hasConsequence(Entity \sqcup Configuration \sqcup Attribute \sqcup$
$AttributeValue \sqcup Behavioural \sqcup Dependency)$

Static fragments are used to describe partial structures of systems. No influences causing change are formalised in static fragments, and thus no new entities, agents and configurations may be introduced (as consequences). Furthermore, no agents may be included as conditions. The formalisation in OWL can be found in appendix B.12.2.

Necessary & Sufficient Static Fragment Restrictions

$\neg(\exists\ hasCondition\ Agent)$
$\neg(\exists\ hasConsequence\ Configuration)$
$\neg(\exists\ hasConsequence\ Entity)$
$\neg(\exists\ hasConsequence\ Influence)$
$\neg(\exists\ hasConsequence\ Agent)$

Process fragments are used to describe the processes which take place in a system. The processes in process fragments may not be caused by agent, which are therefore prohibited in these fragments. The formalisation in OWL is shown in appendix B.12.3.

Necessary & Sufficient Process Fragment Restrictions

$\neg(\exists\ hasCondition\ Agent)$
$\exists\ hasConsequence\ Influence$

Agent fragments describe situations in which an agent may interact with a system. Every model fragment which contains an agent should be an agent fragment. The formalisation in OWL can be found in appendix B.12.4.

Necessary & Sufficient Agent Fragment Restrictions

$\exists\ hasCondition\ Agent$

5.7.2 Scenarios

Scenarios describe situations of the modelled systems which becomes the start node of the state graph. All the ingredients which may be used as conditions in model fragments (table 1), except Model Fragments, may be used in scenarios. The OWL formalisation of the scenario concept can

be found in appendix B.12.5. An OWL formalisation of an example scenario can be found in appendix C.8.

$$\forall\, hasFact(Structural \sqcup Behavioural \sqcup AssumptionType \sqcup Inequality \sqcup PlusMin)$$

5.8 Consequences of Aggregate Formalisation in Models on Reasoning

The formalisation of the aggregates has had consequences for the possibility of using an OWL reasoner to solve QR problems. Finding model fragments which match on a scenario is one of the problems were the OWL reasoner might be useful. The OWL reasoner can classify individuals to classes, so the model fragments would have to be formalised as classes using necessary and sufficient conditions, and the scenarios as individuals. A problem is that the consequences cannot be part of the necessary and sufficient conditions. This is required, as the consequences indicate the individuals which have to be introduced, and are, for that reason, not necessarily present in scenarios. So an ideal formalisation would would separate the conditions of model fragments from the consequences. The classes formalising the conditions would describe general situations in the system, which the specific scenarios could match on.

It is impossible to formalise the conditions of model fragments in classes for two reasons. Firstly, it is not possible to distinghuish between two objects of the same type within a class. For example, it is not possible to specify that that a container x is full and has a relation with container y, and container y has a relation with container z. As OWL does not have the expressiveness to refer to specific individuals in this way, modelling model fragments as classes is impossible. In OWL it is only possible to describe the restriction as: "There is a container which is full, which is connected to a container, which is connected to a container." Such a restriction has multiple interpretations, and is as such unusable. It would even be possible to create a scenario with only 2 containers which would fulfil the restrictions. Secondly, OWL does not have qualified cardinality restrictions, which indicate a restriction that a relation has at least a certain number of targets of a specific type. As a result, it is impossible to ensure that a model fragment incorporates, for example, three containers. This problem will probably be solved in the next iteration of the OWL specification, as the ProtégéOWL editor, and the Racer OWL reasoner have recently added qualified cardinality restriction support.

If these problems were to be solved, it might be possible to introduce the consequences of a model fragment in a scenario using the Semantic Web Rule Language [28]. There would be two problems which would have to be solved. The first problem is that part of the classification task in QR context (see section 4.3.1) is the derivation of inequalities. From existing inequalities, new inequalities can be derived, which make it possible for a scenario to match on new model fragments. It is not possible to do such reasoning using an OWL reasoner. The second problem is that if no more model fragments can be found which match on the scenario, inequalities may be assumed. Again, this allows new model fragments to match on the scenario. This too is an impossible inference for the OWL reasoner to solve. As the formalisation of model fragments as classes is impossible, solutions to these problems have not been researched.

As a result of the impossibility to express model fragments as classes in OWL, the formalisation solution has to use individuals. Another problem is that model fragment can have subclasses. These subclasses contain exactly the same conditions and consequences as its parent, but introduce some new elements. The only way to create subclasses in OWL is from classes. Furthermore, the model fragments can also be incorporated in other model fragments as conditions. The incorporated model fragments are instances of the generic model fragment. This is another reason model fragments have to be formalised as classes, as it is impossible to create instances of instances in OWL. To summerise, the model fragments have to be formalised as classes, but the conditions and consequences as individuals.

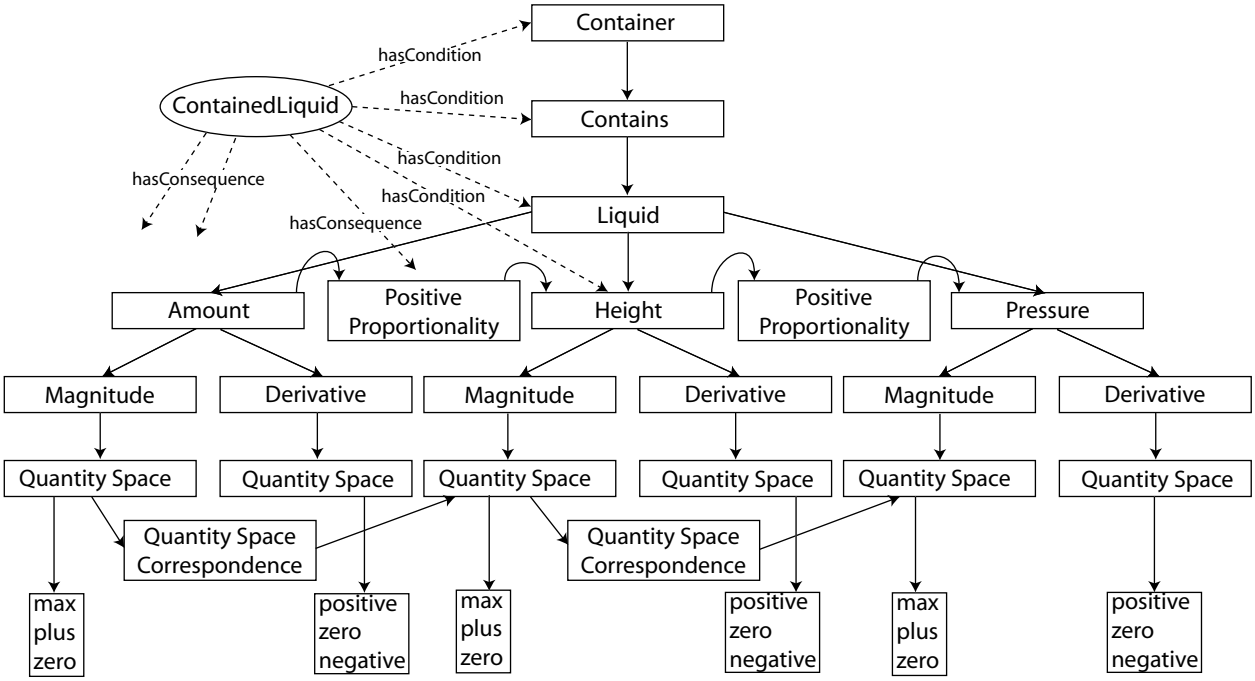

Figure 28: An example of the formalisation of a model fragment. Note that that most of the hasCondition and hasConsequence relations have not been drawn.

This problem can only be solved by treating the model fragment classes as individuals in OWL, thus making domain ontologies OWL Full. The model fragment definitions are classes, so a class hierarchy of model fragments can be created. These classes have hasCondition and hasConsequence relations with instances of the QR ingredients they incorporate. This is ontologically not the most desirable solution, as the conditions in model fragments do not correspond to the conditions in OWL. A further disadvantage is that the individuals which model the model fragments pose no restrictions. Therefore, it is impossible to use an OWL reasoner in any way to classify scenarios on model fragments, as individuals cannot be classified on individuals. Furthermore, when model fragments are reused in other model fragments, the reasoner does not indicate the domain ontology is inconsistent when not all of the conditions or consequences belonging to the incorporated model fragment are mentioned in the incorporating model fragment.

An example of a model fragment formalisation using the OWL Full solution is shown in figure 28. Note that not all the hasCondition and hasConsequence relations have been drawn from the model fragment class to the incorporated individuals. Also the names of the relations have been left out, in order to create an readable figure. The formalisation of this model fragment in OWL can be found in appendix C.7.

5.9 Conclusions

This chapter presented the formalisation of the QR vocabulary in a generic ontology, which is OWL DL. It also described how a qualitative model can be represented in a domain ontology, using the vocabulary defined in the generic ontology. The domain ontologies are in OWL Full, because of the formalisation of model fragments as classes with relations to individuals. A parser has been developed to export the GARP knowledge representation into its OWL counterpart.

The research questions which were posed for this chapter in the introduction are evaluated in the following paragraphs. *"Is the Web Ontology Language expressive enough to capture the QR vocabulary and model formalisations?"* The formalisation of the qualitative vocabulary and models has been succesful in the sense that it is possible to describe a qualitative model and its vocabulary

in OWL. Some problems have persisted however. Not all restrictions which should apply to the qualitative model ingredients can be formalised in OWL. Furthermore, the formalisation of the model fragments is not ideal, due to the limitations in the expressiveness of OWL.

"Which existing general patterns can be used to solve formalisation problems during the development of the ontology, and which reusable solutions are discovered?" The patterns which have been reused were: reification patterns, the property values as a set of individuals pattern, the property values as a set of classes pattern, and the property value as a list pattern. The discovered patterns are: modelling restrictions of reificated classes in the class owning the relation, combining necessary and necessary and sufficient conditions to model restrictions for classes having certain conditions, and modeling "lists" in an ontological manner.

"What conceptual patterns cannot be expressed in OWL?" Some of the patterns which could not be expressed in OWL were: restricting reflexive relations, restricting a relation to a class having an equal cardinality for some property, talking about specific individuals in restrictions, and qualified cardinality restrictions.

What QR problems can an OWL reasoner solve? The OWL reasoner might be used to classify scenarios on model fragments. Because OWL is not expressive enought to formalise model fragments as classes, this goal is not pursued further. There are two other problems which would have to be solved in order to classify scenarios on model fragments. Firstly, valid inequalities have to be derived in order to match a scenario on model fragments in which these inequalities are incorporated as conditions. Secondly, inequalities may be assumed if the value of a magnitude or derivative is not derivable. These problems would be difficult, if not impossible to solve using an OWL reasoner.

6 Conclusions, Discussion and Further Research

The research presented in this thesis has resulted in a generic ontology in which the qualitative reasoning vocabulary has been captured. The domain specific qualitative models are formalised in domain ontologies. Furthermore, a parser has been developed which converts the GARP knowledge representation into its OWL counterpart; a domain ontology. During development of these ontologies existing formalisation patterns have been used. When the qualitative vocabulary could not be captured using these patterns, new patterns were developed. Some restrictions which should apply to qualitative models could not be formalised. These problems clarify the limitations of OWL's expressiveness.

6.1 OWL Patterns

To be able to formalise tertiary relations and store positioning information the QR relations have to be reificated. Two existing reification patterns were compared. The first has an independent object owning the reificated relation, and the reificated relation owning a dependent object. In the second pattern, the reificated relation owns all objects. Both patterns make it possible to model restrictions in the reificated relations, as they are classes. However, this prohibits other classes to use the relation with another range. This problem was solved by developing a new pattern. The restrictions of reificated relations should be described in the classes which use them. This way, every class can use the relation, and a specific range can be specified by each class.

In order to formalise attributes, an existing property value pattern has been used. The possible values of an attribute are formalised as an enumeration of individuals. The result is that there is only one unique individual for each possible attribute value, which is reused in every model fragment. This solution suffices for three reasons. Firstly, the values do not partipate in their own relations. Secondly, have no specific information has to be stored about attribute values (like positioning information). Finally, the intended meaning of attribute values is identical wherever it is used.

The qualitative values of quantity spaces cannot be formalised as a set of individuals, as two values with the same name can have a different value (due to an inequality relation). This makes reuse of the same individual throughout the ontology impossible. This problem was solved by using another existing pattern. Instead of individuals, the qualitative values were formalised as the union of a set of classes. As a result, each quantity space in a model fragment has its own set of instances of the classes defining the qualitative values.

Modeling the qualitative values as classes is not sufficient to fully formalise quantity spaces. The values in a quantity space have a specific ordering which indicates the progression of change of a quantity. In order to formalise this problem existing list patterns could be used. One solution is to use the rdf:List element to formalise a list. A disadvantage is that this would make the ontology OWL Full, as this element may not be used in OWL DL. Another solution is to formalise a list in OWL, which would keep the ontology OWL DL. This has two disadvantages. Firstly, it would make it difficult to deduce if a value belongs to a magnitude or a derivative, as the list has to be traversed. Secondly, a list is not an ontological structure, but a data structure. The ordering problem was solved by attaching quantity spaces to their values using containsQualitativeValue relations, and ordering the qualitative values by using inequality relations. Epistemologically, this makes more sense than using a list, which should be considered bad practice.

To be able to impose restrictions on points beloning to derivatives, and points beloning to magnitudes, without defining them as such, another pattern was developed. The restrictions which have to be imposed are that inequality relations are only valid between points which both belong to either magnitudes or to derivatives. The classes were distinghuised by defining PointBeloningTo-Magnitude and PointBelongingToDerivative. Necessary and sufficient conditions are defined for these classes, stating that they have to belong to either a magnitude or a derivative. This allows points to be classified to either class. Furthermore, the necessary restriction that inequality relations may only target points which belong to a magnitude (or derivative) are defined. The necessary restric-

tions apply once a point has been classified. To summerise, concepts with certain conditions can be restricted by introducing a new class in which necessary, and necessary and sufficient conditions are combined.

6.2 Restriction Formalisation Problems

Some restrictions which should apply to the qualitative vocabulary are impossible to formalise in OWL. The most important problem is the inability to state that the domain and the range of relations should or should not be the same individual. This makes it impossible to prohibit reflexive and non-reflexive relations. This problem influences all the QR relations, and makes it possible to create an illegal model which an OWL reasoner will classify as consistent. The inequality between a magnitude (or derivative) and a value from its quantity space is also affected by this problem. This is caused by the impossibility to specify that the domain of the inequality, the magnitude (or derivative), is the same individual as the magnitude (or derivative) belonging to the range of the inequality, the qualitative value. A different restriction which was impossible to formalise is that quantity space correspondences should only be possible between quantity spaces with the same amount of values (the same cardinality).

6.3 Dealing with Complex Knowledge Structures

The formalisation of model fragments is only partially solved, as it is ontologically imperfect. Every model fragment is formalised as a class, which is epistemologically correct as a model fragment describes a generic situation. It also allows subclasses of model fragments, and the incorporation of model fragments in other model fragments by creating instances. Because it is impossible to distinguish between different objects in OWL restrictions, it was not possible to model the contents of model fragments within the class. In order to create an unambiguously representation of model fragments, their classes are treated as individuals, meaning they have relations with the individuals they contain. This solution makes the domain ontology OWL full, due to the strict typing of OWL DL. A disadvantage is that model fragments which are incorporated in other model fragments are not enforced to correctly contain all the necessary ingredients, as they are described as individuals instad of restrictions.

There are two reasons why it was impossible to formalise model fragments as pure classes. Firstly, OWL does not have qualified cardinality restrictions. Therefore, it is impossible to state, for example, that a model fragment contains two entities of a specific type. Although not in the OWL specification, qualified cardinality restrictions have been added to an editor and a reasoner recently[9]. There is a significant chance that qualified cardinality restrictions will be incorporated in the next iteration of the OWL specification. Secondly, it is not possible to distinguish between to objects of the same type in a model fragment. For this reason, it is not clear to which of the ingredients restrictions would apply. The result would be that model fragments have multiple interpretations.

6.4 Increasing Expressiveness

The research presented in this thesis has uncovered conceptual patterns which are impossible to formalise in OWL. A further study could research the possibility of increasing the expressiveness of OWL by extending the language. The implications of these extension will have consequences for the decidability of the language, which would have to be analysed.

[9] http://protege.stanford.edu/mail_archive/msg17798.html

6.5 OWL Reasoner Solving QR Problems

As the contents of model fragments is formalised using individuals it is not possible to use the OWL reasoner to match scenarios on model fragments. The ideal solution would be to describe the conditions of a model fragment in a class. The consequences would be formalised as the consequence part of a rule, with the condition class as the head. These rules could be formalised using the Semantic Web Rule Language. An OWL reasoner could possibly classify scenarios on the general situation descriptions described by the model fragment conditions. The rule would allow the incorporation of the consequence ingredients in the scenario. However, two problems would still have to be solved. Firstly, the QR reasoner derives valid inequalities from a scenario. This inference is probably impossible to solve using an OWL reasoner. Secondly, the QR reasoner assumes inequalities are true when no more model fragments match a scenario. This too is something an OWL reasoner cannot solve.

6.6 The Qualitative Reasoning Vocabulary Revisited

The development of the ontology has clarified the QR vocabulary. The ingredients have been sorted in semantically related sets, making it easier for new researchers to learn the QR concepts. For instance, the ingredients have been sorted in the groups structural, behavioural, and assumption types. Furthermore, the QR vocabulary has been slightly adapted. The word *magnitude* was introduced to refer to the zero-derivative of a quantity. This makes it easier to explain how proportionalities and influences affect magnitudes and derivatives. It also moves the inequalities from the quantity in GARP to the magnitude in the OWL representation, making it clearer what is meant. A distinction was made between quantity space correspondences and value correspondences, as the meaning of these relations is different. A quantity space correspondence is actually a set of multiple value correspondences.

6.7 Towards Model Sharing and Reuse

Collaboration has become an important approach for learning and working (Computer Supported Collaborative Learning/Work - CSCL/CSCW). When dealing with qualitative models this involves among others sharing and re-using models and model parts. To enable such reusability and sharing of qualitative models, further problems have to be solved. One aspect concerns the development of functionality to load a model expressed in the OWL representation into the GARP engine. This is mostly a technical problem, if the assumption is made that the model described using OWL is a valid model. A second aspect concerns the creation of repositories containing qualitative models and model parts. Methods are needed to search for models in such 'content' repositories. Collaborative working will create versioning problems which will have to be solved. Finally, to actually reuse a model as a part of a model already build, means are needed to support users in consistently merging such knowledge models.

References

[1] Building qualitative models of thermodynamic processes. In *Proc. 3rd Int. Workshop on Qualitative Physics*, Stanford, CA, 1989.

[2] Sean Bechhofer, Frank van Harmelen, Jim Hendler, Ian Horrocks, Deborah L. McGuinness, Peter F. Patel-Schneider, and Lynn Andrea Stein. OWL web ontology language reference. W3C recommendation, World Wide Web Consortium (W3C), February 2004.

[3] Dave Beckett (editor) and Brian McBride (series editor). RDF/XML syntax specification (revised). W3C recommendation, World Wide Web Consortium (W3C), February 2004.

[4] Willem Nico Borst. *Construction of Engineering Ontologies for Knowledge Sharing and Reuse.* PhD thesis, Centre of Telematics and Information Technology, Enschede, Netherlands, 1997.

[5] Ronald J. Brachman and James G. Schmolze. An overview of the KL-ONE knowledge representation system. *Cognitive Science*, 9(2):171–216, 1985.

[6] Tim Bray, Dave Hollander, and Andrew Layman. Namespaces in XML. W3C recommendation, World Wide Web Consortium (W3C), January 1999.

[7] Tim Bray, Jean Paoli, C. M. Sperberg-McQueen, Eve Maler, and François Yergeau. Extensible markup language (XML) 1.0 (third edition). W3C recommendation, World Wide Web Consortium (W3C), February 2004.

[8] Bert Bredeweg. *Expertise in Qualitative Prediction of Behaviour.* PhD thesis, University of Amsterdam, 1992.

[9] Bert Bredeweg and Peter Struss (editors). Current topics in qualitative reasoning. *AI Magazine (special issue)*, 24(4):13–130, Winter 2003.

[10] Dan Brickley (editor), R.V. Guha (editor), and Brian McBride (series editor). RDF vocabulary description language 1.0: RDF Schema. W3C recommendation, World Wide Web Consortium (W3C), February 2004.

[11] Randall Davis, Howard Shrobe, and Peter Szolovits. What is a knowledge representation? *AI Magazine*, 14(1):17–33, Spring 1993.

[12] Johan de Kleer and John Seely Brown. A qualitative physics based on confluences. *Artificial Intelligence*, 24(1-3):7–83, December 1984.

[13] Kenneth D. Forbus. Qualitative process theory. *Artificial Intelligence*, 24(1-3):85–168, December 1984.

[14] Kenneth D. Forbus and Brian Falkenhainer. Compositional modeling: Finding the right model for the job. *Artificial Intelligence*, 51(1-3):95–143, October 1991.

[15] Michael R. Genesereth and Richard E. Fikes. Knowledge interchange format version 3.0 reference manual. Technical report, Knowledge Systems Laboratory, Stanford University, 1992.

[16] Asunción Gómez-Pérez, Mariano Fernández-López, and Oscar Corcho. *Ontological Engineering with examples from the areas of Knowledge Management, e-Commerce and the Semantic Web.* Advanced Information and Knowledge Processing. Springer, London, 2004.

[17] Jan Grant (editor), Dave Beckett (series editor), and Brian McBride (series editor). RDF test cases. W3C recommendation, World Wide Web Consortium (W3C), February.

[18] Thomas R. Gruber. Ontolingua: A mechanism to support portable ontologies. Technical report, Knowledge Systems Laboratoy, University of Stanford, 1992.

[19] Thomas R. Gruber. Towards principles for the design of ontologies used for knowledge sharing. Technical Report KSL 93-04, Knowledge Systems Laboratory, Stanford University, March 1993.

[20] Thomas R. Gruber. A translation approach to portable ontology specifications. *Knowledge Acquisition*, 5(2):199–220, June 1993.

[21] Patrick J. Hayes. *Formal Theories of the Commonsense World*, volume 1 of *Ablex series in Artificial Intelligence*, chapter The Second Naive Physics Manifesto, pages 1–36. Ablex, Norwood, NJ, June 1985.

[22] Patrick J. Hayes. Naive physics I: ontology for liquids. In Jerry R. Hobbs and R. C. Moore, editors, *Formal theories of the commonsense worldve*, pages 71–108. Ablex Publishing Corporation, New Jersey, 1985.

[23] Patrick Hayes (editor) and Brian McBride (series editor). RDF semantics. W3C recommendation, World Wide Web Consortium (W3C), February 2004.

[24] Jeff Heflin. OWL web ontology language use cases and requirements. W3C recommendation, World Wide Web Consortium (W3C), February 2004.

[25] Gertjan van Heijst, Sabina Falasconi, Ameen Abu-Hanna, Guus Schreiber, and Mario Stefanelli. A case study in ontology library contruction. *Artificial Intelligence in Medicine*, 7(3):227–255, June 1995.

[26] Diane Hillmann. Using Dublin Core. DCMI recommended resource, Dublin Core Metadata Initiative, August 2003.

[27] Matthew Horridge, Holger Knublauch, Alan Rector, Robert Stevens, and Chris Wroe. A practical guide to building owl ontologies using the protégé-owl plugin and co-ode tools. edition 1.0. Technical report, The University Of Manchester, August 2004.

[28] Ian Horrocks, Peter F. Patel-Schneider, Harold Boley, Said Tabet, Benjamin Grosof, and Mike Dean. SWRL: A semantic web rule language combining OWL and RuleML version 0.5. W3C member submission, World Wide Web Consortium, May 2004.

[29] Ian Horrocks, Peter F. Patel-Schneider, and Frank van Harmelen. From SHIQ and RDF to OWL: The making of a web ontology language. *Journal of Web Semantics*, 1(1):7–26, December 2003.

[30] Graham Klyne (editor), Jeremy J. Carroll (editor), and Brian McBride (series editor). Resource description framework (RDF): Concepts and abstract syntax. W3C recommendation, World Wide Web Consortium (W3C), February 2004.

[31] Hector J. Levesque and Ronald J. Brachman. *Readings in Knowledge Representation*, chapter A Fundamental Tradeoff in Knowledge Representation and Reasoning (Revised Version), pages 42–70. Morgan Kaufmann, 1985.

[32] Frank Manola (editor), Eric Miller (editor), and Brian McBride (series editor). RDF primer. W3C recommendation, World Wide Web Consortium (W3C), 2004 2004.

[33] John McCarthy. *Semantic Information Processing*, chapter Programs with Common Sense, pages 403–418. MIT Press, Cambridge, Massachutsetts, 1968.

[34] John McCarthy. Circumscription - a form of non-monotonic reasoning. *Artificial Intelligence*, 13(1):27–39, 1980.

[35] Deborah L. McGuinness and Frank van Harmelen. OWL web ontology language overview. W3C recommendation, World Wide Web Consortium (W3C), February 2004.

[36] Allen Newell. The knowledge level. *AI Magazine*, 2(2):1–20;33, Summer 1981.

[37] Natalya F. Noy and Deborah L. McGuinness. Ontology development 101: A guide to creating your first ontology. Technical Report KSL-01-05, Knowledge Systems Lab, Stanford University, March 2001.

[38] Natasha Noy and Alan Rector. Defining n-ary relations on the semantic web. W3C working draft, World Wide Web Consortium (W3C), January 2005.

[39] Peter F. Patel-Schneider, Patrick Hayes, and Ian Horrock. OWL web ontology language semantics and abstract syntax. W3C recommendation, World Wide Web Consortium (W3C), February 2004.

[40] Alan Rector. Representing specified values in OWL: "value partitions" and "value sets". W3C working draft, World Wide Web Consortium (W3C), August 2004.

[41] Paulo Salles and Bert Bredeweg. Qualitative reasoning about population and community ecology. *AI Magazine*, 24(4):77–90, January 2004.

[42] Manfred Schmidt-Schauß. Subsumption in KL-ONE is undecidable. In *Proc. of 1st Conf. on Knowledge Representation and Reasoning*, pages 421–431, 1989.

[43] Guus Schreiber, Hans Akkermans, Anjo Anjewierden, Robert de Hoog, Nigel Shadbolt, Walter van de Velde, and Bob Wielinga. *Knowledge Engineering and Management - The CommonKADS Methodology*. MIT Press, Cambridge, Massachusetts, 2000.

[44] Barry Smith and Chris Welty. Ontology: Towards a new synthesis. In Chris Welty and Barry Smith, editors, *Formal Ontology in Information Systems*, pages iii–ix. ACM Press, October 2001.

[45] Michael K. Smith, Chris Welty, and Deborah L. McGuinness. OWL web ontology language guide. W3C recommendation, World Wide Web Consortium (W3C), February 2004.

[46] John Sowa. *Knowledge Representation - Logical, Philosophical, and Computational Foundations*. Brooks/Cole, Pacific Grove, 2000.

[47] John Sowa. Building, sharing, and merging ontologies. Available at: http://www.jfsowa.com/ontology/ontoshar.htm. Visited on 18-05-2005., 2001. This paper consists of excerpts from previously published articles by John F. Sowa, updated with new material about ongoing projects on ontology and their implications for databases, knowledge bases, and the semantic web.

[48] Rudi Studer, V. Richard Benjamins, and Dieter Fensel. Knowledge engineering: Principles and methods. *Data & Knowledge Engineering*, 25(1-2):161–197, March 1998.

[49] Frank van Harmelen. OWL: an ontology language for the Semantic Web. Presentation available on: http://www.cs.unimaas.nl/SIKSsymp/harmelen.pdf. Visited on: 11 May 2005.

A OWL Patterns

A.1 Relation Reification Pattern 1

```
<owl:Class rdf:ID="RelationOwner">
    <rdfs:subClassOf>
        <owl:Restriction>
            <owl:onProperty rdf:resource="#hasRelation" />
            <owl:allValuesFrom rdf:resource="#Relation" />
        </owl:Restriction>
    </rdfs:subClassOf>
</owl:Class>
<owl:ObjectProperty rdf:ID="hasRelation" />
<owl:Class rdf:ID="Relation">
    <rdfs:subClassOf>
        <owl:Restriction>
            <owl:onProperty rdf:resource="#hasRequiredDependentObject" />
            <owl:allValuesFrom rdf:resource="#RequiredDependentObject" />
        </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
        <owl:Restriction>
            <owl:onProperty rdf:resource="#hasRequiredDependentObject" />
            <owl:cardinality rdf:datatype="&xsd;nonNegativeInteger">
        </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
        <owl:Restriction>
            <owl:onProperty rdf:resource="#hasOptionalDependentObject" />
            <owl:allValuesFrom rdf:resource="#OptionalDependentObject" />
        </owl:Restriction>
    </rdfs:subClassOf>
</owl:Class>
<owl:ObjectProperty rdf:ID="hasRequiredDependentObject" />
<owl:ObjectProperty rdf:ID="hasOptionalDependentObject" />
<owl:Class rdf:ID="RequiredDependentObject" />
<owl:Class rdf:ID="OptionalDependentObject" />
```

A.2 Relation Reification Pattern 2

```
 <owl:Class rdf:ID="Relation">
    <rdfs:subClassOf>
        <owl:Restriction>
            <owl:onProperty rdf:resource="#hasRequiredDependentObject" />
            <owl:allValuesFrom rdf:resource="#RequiredDependentObject" />
        </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
        <owl:Restriction>
            <owl:onProperty rdf:resource="#hasOptionalDependentObject" />
            <owl:allValuesFrom rdf:resource="#OptionalDependentObject" />
        </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
        <owl:Restriction>
            <owl:onProperty rdf:resource="#hasRequiredIndependentObject" />
            <owl:allValuesFrom rdf:resource="#RequiredIndependentObject" />
        </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
        <owl:Restriction>
            <owl:onProperty rdf:resource="#hasRequiredIndependentObject" />
            <owl:cardinality rdf:datatype="&xsd;nonNegativeInteger">1
```

```
            </owl:cardinality>
        </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
    <owl:Restriction>
        <owl:onProperty rdf:resource="#hasOptionalIndependentObject" />
        <owl:allValuesFrom rdf:resource="#OptionalIndependentObject" />
    </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
        <owl:Restriction>
           <owl:onProperty rdf:resource="#hasRequiredIndependentObject2" />
           <owl:allValuesFrom rdf:resource="#RequiredIndependentObject2" />
        </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
        <owl:Restriction>
            <owl:onProperty rdf:resource="#hasRequiredIndependentObject2" />
            <owl:cardinality rdf:resource="&xsd;nonNegativeInteger" />
        </owl:Restriction>
    </rdfs:subClassOf>
</owl:Class>
<owl:ObjectProperty rdf:ID="hasRequiredDependentObject" />
<owl:ObjectProperty rdf:ID="hasOptionalDependentObject" />
<owl:ObjectProperty rdf:ID="hasRequiredIndependentObject">
    <owl:inverseOf rdf:resource="#isRequiredIndependentObjectFor" />
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="hasOptionalIndependentObject">
    <owl:inverseOf rdf:resource="#isOptionalIndependentObjectFor" />
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="hasRequiredIndependentObject2">
    <owl:inverseOf rdf:resource="#isRequiredIndependentObject2For" />
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="isRequiredIndependentObjectFor" />
<owl:ObjectProperty rdf:ID="isRequiredIndependentObject2For" />
<owl:ObjectProperty rdf:ID="isOptionalIndependentObjectFor" />
<owl:Class rdf:ID="RequiredDependentObject" />
<owl:Class rdf:ID="OptionalDependentObject" />
<owl:Class rdf:ID="RequiredIndependentObject" />
<owl:Class rdf:ID="OptionalIndependentObject" />
<owl:Class rdf:ID="RequiredIndependentObject2" />
```

A.3 Reificated Property Restrictions

```
<owl:Class rdf:ID="Person">
    <rdfs:subClassOf>
        <owl:Restriction>
            <owl:onProperty rdf:resource="#hasPlaysInstrumentRelation" />
            <owl:allValuesFrom>
                <owl:Class>
                    <owl:intersectionOf rdf:parseType="Collection">
                        <owl:Class rdf:about="#playsInstrument" />
                        <owl:Restriction>
                                <owl:onProperty rdf:resource="#plays" />
                                <owl:allValuesFrom rdf:resource="#Instrument" />
                        </owl:Restriction>
                    </owl:intersectionOf>
                </owl:Class>
            </owl:allValuesFrom>
        </owl:Restriction>
    </rdfs:subClassOf>
</owl:Class>
<owl:Class rdf:ID="Robot">
```

```
    <rdfs:subClassOf>
        <owl:Restriction>
            <owl:onProperty rdf:resource="#hasPlaysInstrumentRelation" />
            <owl:allValuesFrom>
                <owl:Class>
                    <owl:intersectionOf rdf:parseType="Collection">
                        <owl:Class rdf:about="#playsInstrument" />
                        <owl:Restriction>
                                <owl:onProperty rdf:resource="#plays" />
                                <owl:allValuesFrom rdf:resource="#Instrument" />
                        </owl:Restriction>
                    </owl:intersectionOf>
                </owl:Class>
            </owl:allValuesFrom>
        </owl:Restriction>
    </rdfs:subClassOf>
</owl:Class>
<owl:Class rdf:ID="playsInstrument" />
<owl:Class rdf:ID="Instrument" />
<owl:ObjectProperty rdf:ID="hasPlaysInstrumentRelation" />
<owl:ObjectProperty rdf:ID="plays" />
```

A.4 Property Values Patterns

A.4.1 Property Values as a Set of Individuals

```
<owl:Class rdf:ID="Meat" />
<owl:ObjectProperty rdf:ID="hasReadiness">
    <rdf:type rdf:resource="&owl;FunctionalProperty" />
    <rdfs:domain rdf:resource="#Meat" />
    <rdfs:range rdf:resource="#Readiness" />
</owl:ObjectProperty>
<owl:Class rdf:ID="Readiness">
    <owl:oneOf rdf:parseType="Collection">
        <owl:Thing rdf:about="#Rare" />
        <owl:Thing rdf:about="#MediumRare" />
        <owl:Thing rdf:about="#Medium" />
        <owl:Thing rdf:about="#WellDone" />
    </owl:oneOf>
</owl:Class>
<Readiness rdf:ID="Rare" />
<Readiness rdf:ID="MediumRare" />
<Readiness rdf:ID="Medium" />
<Readiness rdf:ID="WellDone" />
<owl:AllDifferent>
    <owl:distinctMembers rdf:parseType="Collection">
        <owl:Thing rdf:about="#Rare" />
        <owl:Thing rdf:about="#MediumRare" />
        <owl:Thing rdf:about="#Medium" />
        <owl:Thing rdf:about="#WellDone" />
    </owl:distinctMembers>
</owl:AllDifferent>
```

A.4.2 Property Values as a Set of Classes

```
<owl:Class rdf:ID="Meat" />
<owl:ObjectProperty rdf:ID="hasReadiness">
    <rdf:type rdf:resource="&owl;FunctionalProperty" />
    <rdfs:domain rdf:resource="#Meat" />
    <rdfs:range rdf:resource="#Readiness" />
</owl:ObjectProperty>
<owl:Class rdf:ID="Readiness">
    <owl:unionOf rdf:parseType="Collection">
```

```
        <owl:Class rdf:about="#Rare" />
        <owl:Class rdf:about="#MediumRare" />
        <owl:Class rdf:about="#Medium" />
        <owl:Class rdf:about="#WellDone" />
    </owl:unionOf>
</owl:Class>
<owl:Class rdf:ID="Rare">
    <rdfs:subClassOf rdf:resource="#Readiness" />
    <owl:disjointWith rdf:resource="#MediumRare" />
    <owl:disjointWith rdf:resource="#Medium" />
    <owl:disjointWith rdf:resource="#WellDone" />
</owl:Class>
<owl:Class rdf:ID="MediumRare">
    <rdfs:subClassOf rdf:resource="#Readiness" />
    <owl:disjointWith rdf:resource="#Rare" />
    <owl:disjointWith rdf:resource="#Medium" />
    <owl:disjointWith rdf:resource="#WellDone" />
</owl:Class>
<owl:Class rdf:ID="Medium">
    <rdfs:subClassOf rdf:resource="#Readiness" />
    <owl:disjointWith rdf:resource="#Rare" />
    <owl:disjointWith rdf:resource="#MediumRare" />
    <owl:disjointWith rdf:resource="#WellDone" />
</owl:Class>
<owl:Class rdf:ID="WellDone">
    <rdfs:subClassOf rdf:resource="#Readiness" />
    <owl:disjointWith rdf:resource="#Rare" />
    <owl:disjointWith rdf:resource="#MediumRare" />
    <owl:disjointWith rdf:resource="#Medium" />
</owl:Class>
<MediumRare rdf:ID="WellDone_1" />
<Meat rdf:ID="SirloinSteak">
    <hasReadiness rdf:resource="#WellDone_1" />
</Meat>
```

A.4.3 A List as a Property Value

```
<owl:Class rdf:ID="ArgumentList" />
<owl:ObjectProperty rdf:ID="RestOfList">
    <rdf:type rdf:resource="&owl;FunctionalProperty" />
    <rdfs:domain rdf:resource="#ArgumentList" />
    <rdfs:range rdf:resource="#ArgumentList" />
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="hasContents">
<rdf:type rdf:resource="&owl;FunctionalProperty" />
    <rdfs:domain rdf:resource="#ArgumentList" />
    <rdfs:range rdf:resource="&owl;Thing" />
</owl:ObjectProperty>
<ArgumentList rdf:ID="EmptyList" />
<owl:Class rdf:ID="MeatReadiness" />
<owl:Class rdf:ID="MeatReadinessValue" />
<owl:Class rdf:ID="Rare">
    <rdfs:subClassOf rdf:resource="#MeatReadiness" />
    <owl:disjointWith rdf:resource="#MediumRare" />
    <owl:disjointWith rdf:resource="#Medium" />
    <owl:disjointWith rdf:resource="#WellDone" />
</owl:Class>
<owl:Class rdf:ID="MediumRare">
    <rdfs:subClassOf rdf:resource="#MeatReadiness" />
    <owl:disjointWith rdf:resource="#Rare" />
    <owl:disjointWith rdf:resource="#Medium" />
    <owl:disjointWith rdf:resource="#WellDone" />
</owl:Class>
```

```
<owl:Class rdf:ID="Medium">
    <rdfs:subClassOf rdf:resource="#MeatReadiness" />
    <owl:disjointWith rdf:resource="#Rare" />
    <owl:disjointWith rdf:resource="#MediumRare" />
    <owl:disjointWith rdf:resource="#WellDone" />
</owl:Class>
<owl:Class rdf:ID="WellDone">
    <rdfs:subClassOf rdf:resource="#MeatReadiness" />
    <owl:disjointWith rdf:resource="#Rare" />
    <owl:disjointWith rdf:resource="#MediumRare" />
    <owl:disjointWith rdf:resource="#Medium" />
</owl:Class>
<Rare rdf:ID="Rare1"/>
<MediumRare rdf:ID="MediumRare1" />
<Medium rdf:ID="Medium1" />
<WellDone rdf:ID="WellDone1" />
<owl:ObjectProperty rdf:ID="hasReadinessSequence">
    <rdfs:domain rdf:resource="#MeatReadiness" />
    <rdfs:range  rdf:resource="#ArgumentList" />
</owl:ObjectProperty>
<MeatReadiness rdf:ID="SirloinSteakReadiness">
    <hasReadinessSequence>
        <ArgumentList rdf:ID="List1">
            <hasContents rdf:resource="#Rare1" />
            <RestOfList>
                <ArgumentList rdf:ID="List2">
                    <hasContents rdf:resource="#MediumRare1" />
                    <RestOfList>
                        <ArgumentList rdf:ID="List3">
                            <hasContents rdf:resource="#Medium1" />
                            <RestOfList>
                                <ArgumentList rdf:ID="List4">
                                    <hasContents rdf:resource="#WellDone1" />
                                    <RestOfList rdf:resource="#EmptyList" />
                                </ArgumentList>
                            </RestOfList>
                        </ArgumentList>
                    </RestOfList>
                </ArgumentList>
            </RestOfList>
        </ArgumentList>
    </hasReadinessSequence>
</MeatReadiness>
```

A.4.4 An Ontological Ordering of Values

B Qualitative Reasoning Vocabulary

B.1 The Qualitative Vocabulary Hierarchy

```
<owl:Class rdf:ID="QualitativeModelIngredient">
    <rdfs:label xml:lang="en">Qualitative Model Ingredient</rdfs:label>
    <rdfs:comment xml:lang="en">
        Qualitative Model Ingredients are either Building Blocks or
        Aggregates.
    </rdfs:comment>
    <owl:unionOf rdf:parseType="Collection">
        <owl:Class rdf:about="#BuildingBlock" />
        <owl:Class rdf:about="#Aggregate" />
    </owl:unionOf>
```

```
</owl:Class>
<owl:Class rdf:ID="BuildingBlock">
    <rdfs:label xml:lang="en">Building Block</rdfs:label>
    <rdfs:comment xml:lang="en">
        Building Blocks are single model ingredients.
    </rdfs:comment>
    <rdfs:subClassOf rdf:resource="#QualitativeModelIngredient" />
    <owl:disjointWith rdf:resource="#Aggregate" />
</owl:Class>
<owl:Class rdf:ID="Aggregate">
    <rdfs:label xml:lang="en">Aggregate</rdfs:label>
    <rdfs:comment xml:lang="en">
        Aggregates are composed out of multiple Building Blocks.
    </rdfs:comment>
    <rdfs:subClassOf rdf:resource="#QualitativeModelIngredient" />
    <owl:disjointWith rdf:resource="#BuildingBlock" />
</owl:Class>
<owl:Class rdf:ID="Structural">
    <rdfs:label xml:lang="en">Structural</rdfs:label>
    <rdfs:comment xml:lang="en">
        Structurals describe the structure of a modeled system.
    </rdfs:comment>
    <rdfs:subClassOf rdf:resource="#BuildingBlock" />
    <owl:disjointWith rdf:resource="#AssumptionType" />
    <owl:disjointWith rdf:resource="#Behavioural" />
</owl:Class>
<owl:Class rdf:ID="AssumptionType">
    <rdfs:label xml:lang="en">Assumption Type</rdfs:label>
    <rdfs:comment xml:lang="en">
        Assumption Type is the set of possible Assumptions types.
    </rdfs:comment>
    <rdfs:subClassOf rdf:resource="#BuildingBlock" />
    <owl:disjointWith rdf:resource="#Structural" />
    <owl:disjointWith rdf:resource="#Behavioural" />
</owl:Class>
<owl:Class rdf:ID="Behavioural">
    <rdfs:label xml:lang="en">Behavioural</rdfs:label>
    <rdfs:comment xml:lang="en">
        Behavioural ingredients describe the behavioural aspects of a
        modeled system.
    </rdfs:comment>
    <rdfs:subClassOf rdf:resource="#BuildingBlock" />
    <owl:disjointWith rdf:resource="#AssumptionType" />
    <owl:disjointWith rdf:resource="#Structural" />
</owl:Class>
<owl:Class rdf:ID="Dependency">
    <rdfs:label xml:lang="en">Dependency</rdfs:label>
    <rdfs:comment xml:lang="en">
        Dependencies describe the behavioural relations between behavioural
        ingredients.
    </rdfs:comment>
    <rdfs:subClassOf rdf:resource="#Behavioural" />
    <owl:disjointWith rdf:resource="#Derivative" />
    <owl:disjointWith rdf:resource="#Magnitude" />
    <owl:disjointWith rdf:resource="#QualitativeValue" />
    <owl:disjointWith rdf:resource="#Quantity" />
    <owl:disjointWith rdf:resource="#QuantitySpace" />
</owl:Class>
<!-- Mathematics: A number assigned to a quantity so that it may be compared
to other quantities. -->
<owl:Class rdf:ID="Mathematical">
    <rdfs:label xml:lang="en">Mathematical</rdfs:label>
    <rdfs:comment xml:lang="en">
```

```
        Mathematical dependencies describe the mathematical relations
        between behavioural ingredients.
    </rdfs:comment>
    <rdfs:subClassOf rdf:resource="#Dependency" />
    <owl:disjointWith rdf:resource="#Correspondence" />
    <owl:disjointWith rdf:resource="#CausalDependency" />
</owl:Class>
<owl:Class rdf:ID="Entity">
    <rdfs:label xml:lang="en">Entity</rdfs:label>
    <rdfs:comment xml:lang="en">
        Entities are the structural ingredients a system is composed of.
    </rdfs:comment>
    <rdfs:subClassOf rdf:resource="#Structural" />
    <owl:disjointWith rdf:resource="#Agent" />
    <owl:disjointWith rdf:resource="#Configuration" />
    <owl:disjointWith rdf:resource="#Attribute" />
    <owl:disjointWith rdf:resource="#AttributeValue" />
</owl:Class>
<owl:Class rdf:ID="Agent">
    <rdfs:label xml:lang="en">Agent</rdfs:label>
    <rdfs:comment xml:lang="en">
        Agents are not part of the inherent structure of the system, but can
        influence the system.
    </rdfs:comment>
    <rdfs:subClassOf rdf:resource="#Structural" />
    <owl:disjointWith rdf:resource="#Entity" />
    <owl:disjointWith rdf:resource="#Configuration" />
    <owl:disjointWith rdf:resource="#Attribute" />
    <owl:disjointWith rdf:resource="#AttributeValue" />
</owl:Class>
<owl:Class rdf:ID="Configuration">
    <rdfs:label xml:lang="en">Configuration</rdfs:label>
    <rdfs:comment xml:lang="en">
        Configurations describe the relations between structural
        ingredients.
    </rdfs:comment>
    <rdfs:subClassOf rdf:resource="#Structural" />
    <owl:disjointWith rdf:resource="#Agent" />
    <owl:disjointWith rdf:resource="#Entity" />
    <owl:disjointWith rdf:resource="#Attribute" />
    <owl:disjointWith rdf:resource="#AttributeValue" />
</owl:Class>
<owl:Class rdf:ID="Attribute">
    <rdfs:label xml:lang="en">Attribute</rdfs:label>
    <rdfs:comment xml:lang="en">
        Attributes describe the properties of entities and agents.
    </rdfs:comment>
    <rdfs:subClassOf rdf:resource="#Structural" />
    <owl:disjointWith rdf:resource="#Agent" />
    <owl:disjointWith rdf:resource="#Configuration" />
    <owl:disjointWith rdf:resource="#Entity" />
    <owl:disjointWith rdf:resource="#AttributeValue" />
</owl:Class>
<owl:Class rdf:ID="AttributeValue">
    <rdfs:label xml:lang="en">Attribute Value</rdfs:label>
    <rdfs:comment xml:lang="en">
        AttributeValues are the possible values of attributes.
    </rdfs:comment>
    <rdfs:subClassOf rdf:resource="#Structural" />
    <owl:disjointWith rdf:resource="#Agent" />
    <owl:disjointWith rdf:resource="#Configuration" />
    <owl:disjointWith rdf:resource="#Attribute" />
    <owl:disjointWith rdf:resource="#Entity" />
```

```
</owl:Class>
<owl:Class rdf:ID="Quantity">
    <rdfs:label xml:lang="en">Quantity</rdfs:label>
    <rdfs:comment xml:lang="en">
        Quantities are the changable features of an entity of agent.
    </rdfs:comment>
    <rdfs:subClassOf rdf:resource="#Behavioural" />
    <owl:disjointWith rdf:resource="#Magnitude" />
    <owl:disjointWith rdf:resource="#Derivative" />
    <owl:disjointWith rdf:resource="#QuantitySpace" />
    <owl:disjointWith rdf:resource="#QualitativeValue" />
    <owl:disjointWith rdf:resource="#Dependency" />
</owl:Class>
<owl:Class rdf:ID="Assumption">
    <rdfs:label xml:lang="en">Assumption</rdfs:label>
    <rdfs:comment xml:lang="en">
        An assumption is an assertion that something is true.
    </rdfs:comment>
    <rdfs:subClassOf rdf:resource="#AssumptionType" />
</owl:Class>
<owl:Class rdf:ID="Magnitude">
    <rdfs:label xml:lang="en">Magnitude</rdfs:label>
    <rdfs:comment xml:lang="en">
        The magnitude is the zero-derivative of a quantity. A value can be
        assigned to it so it can be compared to other quantities.
    </rdfs:comment>
    <rdfs:subClassOf rdf:resource="#Behavioural" />
    <owl:disjointWith rdf:resource="#Quantity" />
    <owl:disjointWith rdf:resource="#Derivative" />
    <owl:disjointWith rdf:resource="#QuantitySpace" />
    <owl:disjointWith rdf:resource="#QualitativeValue" />
    <owl:disjointWith rdf:resource="#Dependency" />
</owl:Class>
<owl:Class rdf:ID="Derivative">
    <rdfs:label xml:lang="en">Derivative</rdfs:label>
    <rdfs:comment xml:lang="en">
        The derivative is the first derivative of a quantity, indicating if
        the quantity is increasing, decreasing or stable. A value can be
        assigned to it so it can be compared to other quantities.
    </rdfs:comment>
    <rdfs:subClassOf rdf:resource="#Behavioural" />
    <owl:disjointWith rdf:resource="#Magnitude" />
    <owl:disjointWith rdf:resource="#Quantity" />
    <owl:disjointWith rdf:resource="#QuantitySpace" />
    <owl:disjointWith rdf:resource="#QualitativeValue" />
    <owl:disjointWith rdf:resource="#Dependency" />
</owl:Class>
<owl:Class rdf:ID="QuantitySpace">
    <rdfs:label xml:lang="en">Quantity Space</rdfs:label>
    <rdfs:comment xml:lang="en">
        A quantity space is a list of possible values the magnitude or
        derivative of a quantity can become.
    </rdfs:comment>
    <rdfs:subClassOf rdf:resource="#Behavioural" />
    <owl:disjointWith rdf:resource="#Magnitude" />
    <owl:disjointWith rdf:resource="#Derivative" />
    <owl:disjointWith rdf:resource="#Quantity" />
    <owl:disjointWith rdf:resource="#QualitativeValue" />
    <owl:disjointWith rdf:resource="#Dependency" />
</owl:Class>
<owl:Class rdf:ID="QualitativeValue">
    <rdfs:label xml:lang="en">Qualitative Value</rdfs:label>
    <rdfs:comment xml:lang="en">
```

```
        A qualitative value is either a point or interval which can become
        the value of the magnitude or derivative of a quantity.
    </rdfs:comment>
    <rdfs:subClassOf rdf:resource="#Behavioural" />
    <owl:disjointWith rdf:resource="#Magnitude" />
    <owl:disjointWith rdf:resource="#Derivative" />
    <owl:disjointWith rdf:resource="#Quantity" />
    <owl:disjointWith rdf:resource="#QuantitySpace" />
    <owl:disjointWith rdf:resource="#Dependency" />
</owl:Class>
<owl:Class rdf:ID="Point">
    <rdfs:label xml:lang="en">Point</rdfs:label>
    <rdfs:comment xml:lang="en">
        A point is a qualitative value.
    </rdfs:comment>
    <rdfs:subClassOf rdf:resource="#QualitativeValue" />
    <owl:disjointWith rdf:resource="#Interval" />
</owl:Class>
<Point rdf:ID="Zero">
    <rdfs:label>Zero</rdfs:label>
</Point>
<owl:Class rdf:ID="Interval">
    <rdfs:label xml:lang="en">Point</rdfs:label>
    <rdfs:comment xml:lang="en">
        An interval is a qualitative value.
    </rdfs:comment>
    <rdfs:subClassOf rdf:resource="#QualitativeValue" />
    <owl:disjointWith rdf:resource="#Point" />
</owl:Class>
<owl:Class rdf:ID="CausalDependency">
    <rdfs:label xml:lang="en">Causal Dependency</rdfs:label>
    <rdfs:comment xml:lang="en">
        Causal dependencies are dependencies indicate which quantities are
        proportional, and which ones influence each other.
    </rdfs:comment>
    <rdfs:subClassOf rdf:resource="#Dependency" />
    <owl:disjointWith rdf:resource="#Correspondence" />
    <owl:disjointWith rdf:resource="#Mathematical" />
</owl:Class>
<owl:Class rdf:ID="Proportionality">
    <rdfs:label xml:lang="en">Proportionality</rdfs:label>
    <rdfs:comment xml:lang="en">
        Proportionalities indicate which quantity changes if the other
        changes.
    </rdfs:comment>
    <rdfs:subClassOf rdf:resource="#CausalDependency" />
    <owl:disjointWith rdf:resource="#Influence" />
</owl:Class>
<owl:Class rdf:ID="PositiveProportionality">
    <rdfs:label xml:lang="en">Positive Proportionality</rdfs:label>
    <rdfs:comment xml:lang="en">
        A positive proportionality indicates that the target quantity
        increases if the origin quantity increases, and decreases if the
        origin quantity descreases.
    </rdfs:comment>
    <rdfs:subClassOf rdf:resource="#Proportionality" />
    <owl:disjointWith rdf:resource="#NegativeProportionality" />
</owl:Class>
<owl:Class rdf:ID="NegativeProportionality">
    <rdfs:label xml:lang="en">Negative Proportionality</rdfs:label>
    <rdfs:comment xml:lang="en">
        A negative proportionality indicates that the target quantity
        decreases if the origin quantity increases, and increases if the
```

```
        origin quantity decreases.
    </rdfs:comment>
    <rdfs:subClassOf rdf:resource="#Proportionality" />
    <owl:disjointWith rdf:resource="#PositiveProportionality" />
</owl:Class>
<owl:Class rdf:ID="Influence">
    <rdfs:label xml:lang="en">Influence</rdfs:label>
    <rdfs:comment xml:lang="en">
        Influences indicate that the target quantity changes if magnitude of
        the origin quantity is not zero.
    </rdfs:comment>
    <rdfs:subClassOf rdf:resource="#CausalDependency" />
    <owl:disjointWith rdf:resource="#Proportionality" />
</owl:Class>
<owl:Class rdf:ID="PositiveInfluence">
    <rdfs:label xml:lang="en">Positive Influence</rdfs:label>
    <rdfs:comment xml:lang="en">
        A positive influence indicates that the target quantity increases if
        the magnitude of the origin quantity is greater than zero, and
        decreases if it less than zero.
    </rdfs:comment>
    <rdfs:subClassOf rdf:resource="#Influence" />
    <owl:disjointWith rdf:resource="#NegativeInfluence" />
</owl:Class>
<owl:Class rdf:ID="NegativeInfluence">
    <rdfs:label xml:lang="en">Negative Influence</rdfs:label>
    <rdfs:comment xml:lang="en">
        A negative influence indicates that the target quantity decreases if
        the magnitude of the origin quantity is greater than zero, and
        increases if it less than zero.
    </rdfs:comment>
    <rdfs:subClassOf rdf:resource="#Influence" />
    <owl:disjointWith rdf:resource="#PositiveInfluence" />
</owl:Class>
<owl:Class rdf:ID="Correspondence">
    <rdfs:label xml:lang="en">Correspondence</rdfs:label>
    <rdfs:comment xml:lang="en">
        Correspondences specify that values occur simultaniously.
    </rdfs:comment>
    <rdfs:subClassOf rdf:resource="#Dependency" />
    <owl:disjointWith rdf:resource="#Mathematical" />
    <owl:disjointWith rdf:resource="#CausalDependency" />
</owl:Class>
<owl:Class rdf:ID="ValueCorrespondence">
    <rdfs:label xml:lang="en">Value Correspondence</rdfs:label>
    <rdfs:comment xml:lang="en">
        A Value Correspondence specifies that two values occur simultaniously.
    </rdfs:comment>
    <rdfs:subClassOf rdf:resource="#Correspondence" />
    <owl:disjointWith rdf:resource="#QuantitySpaceCorrespondence" />
</owl:Class>
<owl:Class rdf:ID="DirectedValueCorrespondence">
    <rdfs:label xml:lang="en">Directed Value Correspondence</rdfs:label>
    <rdfs:comment xml:lang="en">
        A Directed Value Correspondence specifies that if a origin magnitude
        or derivative has a specific value, the target value may be
        inferred.
    </rdfs:comment>
    <rdfs:subClassOf rdf:resource="#ValueCorrespondence" />
    <owl:disjointWith rdf:resource="#UndirectedValueCorrespondence" />
</owl:Class>
<owl:Class rdf:ID="UndirectedValueCorrespondence">
    <rdfs:label xml:lang="en">Undirected Value Correspondence</rdfs:label>
```

```
    <rdfs:comment xml:lang="en">
        An Undirected Value Correspondence specifies that if the either the
        origin magnitude or derivative has a specific value, the target
        value may be inferred, and vice versa.
    </rdfs:comment>
    <rdfs:subClassOf rdf:resource="#ValueCorrespondence" />
    <owl:disjointWith rdf:resource="#DirectedValueCorrespondence" />
</owl:Class>
<owl:Class rdf:ID="QuantitySpaceCorrespondence">
    <rdfs:label xml:lang="en">Quantity Space Correspondence</rdfs:label>
    <rdfs:comment xml:lang="en">
        Quantity Space Correspondences are used to denote that the values of
        two quantity spaces correspond to each other.
    </rdfs:comment>
    <rdfs:subClassOf rdf:resource="#Correspondence" />
    <owl:disjointWith rdf:resource="#ValueCorrespondence" />
</owl:Class>
<owl:Class rdf:ID="DirectedQuantitySpaceCorrespondence">
    <rdfs:label xml:lang="en">Directed Quantity Space Correspondence</rdfs:label>
    <rdfs:comment xml:lang="en">
        Directed Quantity Space Correspondences are used to denote that the
        values of the origin quantity space have a directed correspondence
        to the values in the target quantity space.
    </rdfs:comment>
    <rdfs:subClassOf rdf:resource="#QuantitySpaceCorrespondence" />
    <owl:disjointWith rdf:resource="#UndirectedQuantitySpaceCorrespondence" />
</owl:Class>
<owl:Class rdf:ID="UndirectedQuantitySpaceCorrespondence">
    <rdfs:label xml:lang="en">Undirected Quantity Space Correspondence</rdfs:label>
    <rdfs:comment xml:lang="en">
        Undirected Quantity Space Correspondences are used to denote that
        the values of the origin quantity spaces have an undirected
        correspondence to the values in the target quantity space.
    </rdfs:comment>
    <rdfs:subClassOf rdf:resource="#QuantitySpaceCorrespondence" />
    <owl:disjointWith rdf:resource="#DirectedQuantitySpaceCorrespondence" />
</owl:Class>
<owl:Class rdf:ID="Inequality">
    <rdfs:label xml:lang="en">Inequality</rdfs:label>
    <rdfs:comment xml:lang="en">
    Inequalities are used to indicate the difference or equality between values.
    </rdfs:comment>
    <rdfs:subClassOf rdf:resource="#Mathematical" />
    <owl:disjointWith rdf:resource="#PlusMin" />
</owl:Class>
<owl:Class rdf:ID="SmallerThan">
    <rdfs:label xml:lang="en">Smaller Than</rdfs:label>
    <rdfs:comment xml:lang="en">
    A smaller than relation indicates that the origin value is smaller than the target value.
    </rdfs:comment>
    <rdfs:subClassOf rdf:resource="#Inequality" />
    <owl:disjointWith rdf:resource="#SmallerOrEqualTo" />
    <owl:disjointWith rdf:resource="#EqualTo" />
    <owl:disjointWith rdf:resource="#GreaterOrEqualTo" />
    <owl:disjointWith rdf:resource="#GreaterThan" />
    <owl:disjointWith rdf:resource="#hasValue" />
</owl:Class>
<owl:Class rdf:ID="SmallerOrEqualTo">
    <rdfs:label xml:lang="en">Smaller Or Equal To</rdfs:label>
    <rdfs:comment xml:lang="en">
    A smaller or equal to relation indicates that the origin value is smaller or equal to the target value.
    </rdfs:comment>
    <rdfs:subClassOf rdf:resource="#Inequality" />
```

```
    <owl:disjointWith rdf:resource="#SmallerThan" />
    <owl:disjointWith rdf:resource="#EqualTo" />
    <owl:disjointWith rdf:resource="#GreaterOrEqualTo" />
    <owl:disjointWith rdf:resource="#GreaterThan" />
    <owl:disjointWith rdf:resource="#hasValue" />
</owl:Class>
<owl:Class rdf:ID="EqualTo">
    <rdfs:label xml:lang="en">Equal To</rdfs:label>
    <rdfs:comment xml:lang="en">
    An equality relation indicates that the origin value is equal to the target value.
    </rdfs:comment>
    <rdfs:subClassOf rdf:resource="#Inequality" />
    <owl:disjointWith rdf:resource="#SmallerThan" />
    <owl:disjointWith rdf:resource="#SmallerOrEqualTo" />
    <owl:disjointWith rdf:resource="#GreaterOrEqualTo" />
    <owl:disjointWith rdf:resource="#GreaterThan" />
    <owl:disjointWith rdf:resource="#hasValue" />
</owl:Class>
<owl:Class rdf:ID="GreaterOrEqualTo">
    <rdfs:label xml:lang="en">Greater Or Equal To</rdfs:label>
    <rdfs:comment xml:lang="en">
    A greater or equal to relation indicates that the origin value is greater or equal to the target value.
    </rdfs:comment>
    <rdfs:subClassOf rdf:resource="#Inequality" />
    <owl:disjointWith rdf:resource="#SmallerThan" />
    <owl:disjointWith rdf:resource="#SmallerOrEqualTo" />
    <owl:disjointWith rdf:resource="#EqualTo" />
    <owl:disjointWith rdf:resource="#GreaterThan" />
    <owl:disjointWith rdf:resource="#hasValue" />
</owl:Class>
<owl:Class rdf:ID="GreaterThan">
    <rdfs:label xml:lang="en">Greater Than</rdfs:label>
    <rdfs:comment xml:lang="en">
    A greater than relation indicates that the origin value is greater than the target value.
    </rdfs:comment>
    <rdfs:subClassOf rdf:resource="#Inequality" />
    <owl:disjointWith rdf:resource="#SmallerThan" />
    <owl:disjointWith rdf:resource="#SmallerOrEqualTo" />
    <owl:disjointWith rdf:resource="#EqualTo" />
    <owl:disjointWith rdf:resource="#GreaterOrEqualTo" />
    <owl:disjointWith rdf:resource="#hasValue" />
</owl:Class>
<owl:Class rdf:ID="hasValue">
    <rdfs:label xml:lang="en">hasValue</rdfs:label>
    <rdfs:comment xml:lang="en">
        hasValue relations indicate a magnitude or derivative have a certain value.
    </rdfs:comment>
    <rdfs:subClassOf rdf:resource="#Inequality" />
    <owl:disjointWith rdf:resource="#SmallerThan" />
    <owl:disjointWith rdf:resource="#SmallerOrEqualTo" />
    <owl:disjointWith rdf:resource="#EqualTo" />
    <owl:disjointWith rdf:resource="#GreaterOrEqualTo" />
    <owl:disjointWith rdf:resource="#GreaterThan" />
</owl:Class>
<owl:Class rdf:ID="PlusMin">
    <rdfs:label xml:lang="en">Plus/Min</rdfs:label>
    <rdfs:comment xml:lang="en">
    PlusMin is the set of addition and substraction relations. The relation
    itself can have multiple inequality relations.
    </rdfs:comment>
    <rdfs:subClassOf rdf:resource="#Mathematical" />
    <owl:disjointWith rdf:resource="#Inequality" />
</owl:Class>
```

```
<owl:Class rdf:ID="Plus">
    <rdfs:label xml:lang="en">Plus</rdfs:label>
    <rdfs:comment xml:lang="en">
    Plus is addition of the origin and target value and can have multiple
    inequalities.
    </rdfs:comment>
    <rdfs:subClassOf rdf:resource="#PlusMin" />
    <owl:disjointWith rdf:resource="#Min" />
</owl:Class>
<owl:Class rdf:ID="Min">
    <rdfs:label xml:lang="en">Plus/Min</rdfs:label>
    <rdfs:comment xml:lang="en">
    Min is substraction of the origin and target value and can have multiple
    inequalities.
    </rdfs:comment>
    <rdfs:subClassOf rdf:resource="#PlusMin" />
    <owl:disjointWith rdf:resource="#Plus" />
</owl:Class>
<owl:Class rdf:ID="ModelFragment">
    <rdfs:label xml:lang="en">Model Fragment</rdfs:label>
    <rdfs:comment xml:lang="en">
    A model fragment describe parts of the structure and behaviour of a
    system and is composed of multiple building blocks.
    </rdfs:comment>
    <rdfs:subClassOf rdf:resource="#Aggregate" />
    <owl:disjointWith rdf:resource="#Scenario" />
</owl:Class>
<owl:Class rdf:ID="StaticFragment">
    <rdfs:label xml:lang="en">Static Fragment</rdfs:label>
    <rdfs:comment xml:lang="en">
    A static fragment is a model fragment without influences or agents.
    </rdfs:comment>
    <owl:disjointWith rdf:resource="#ProcessFragment" />
    <owl:disjointWith rdf:resource="#AgentFragment" />
</owl:Class>
<owl:Class rdf:ID="ProcessFragment">
    <rdfs:label xml:lang="en">Process Fragment</rdfs:label>
    <rdfs:comment xml:lang="en">
    A process fragment is a model fragment which models a process, but does
    not contain agents.
    </rdfs:comment>
    <owl:disjointWith rdf:resource="#StaticFragment" />
    <owl:disjointWith rdf:resource="#AgentFragment" />
</owl:Class>
<owl:Class rdf:ID="AgentFragment">
    <rdfs:label xml:lang="en">Agent Fragment</rdfs:label>
    <rdfs:comment xml:lang="en">
    An agent fragment is a model fragment which contains an agent.
    </rdfs:comment>
    <owl:disjointWith rdf:resource="#ProcessFragment" />
    <owl:disjointWith rdf:resource="#StaticFragment" />
</owl:Class>
<owl:Class rdf:ID="Scenario">
    <rdfs:label xml:lang="en">Scenario</rdfs:label>
    <rdfs:comment xml:lang="en">
    A scenario describes a situation of the system which becomes the
    start node of the state graph.
    </rdfs:comment>
    <rdfs:subClassOf rdf:resource="#Aggregate" />
    <owl:disjointWith rdf:resource="#ModelFragment" />
</owl:Class>
```

B.2 Qualitative Model Ingredient Restrictions

```
<owl:Class rdf:about="#QualitativeModelIngredient">
    <rdfs:subClassOf>
        <owl:Restriction>
            <owl:onProperty>
                <owl:DatatypeProperty rdf:ID="has_xposition_on_screen"/>
            </owl:onProperty>
            <owl:maxCardinality rdf:datatype="&xsd;nonNegativeInteger">1
            </owl:maxCardinality>
        </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
        <owl:Restriction>
            <owl:onProperty>
                <owl:DatatypeProperty rdf:ID="has_yposition_on_screen"/>
            </owl:onProperty>
            <owl:maxCardinality rdf:datatype="&xsd;nonNegativeInteger">1
            </owl:maxCardinality>
        </owl:Restriction>
    </rdfs:subClassOf>
</owl:Class>
```

B.3 Entity and Agent Restrictions

```
<owl:Class rdf:about="#Entity">
    <rdfs:subClassOf>
        <owl:Restriction>
            <owl:onProperty rdf:resource="#hasAttribute" />
            <owl:allValuesFrom rdf:resource="#Attribute" />
        </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
        <owl:Restriction>
            <owl:onProperty rdf:resource="#hasQuantity" />
            <owl:allValuesFrom rdf:resource="#Quantity" />
        </owl:Restriction>
    </rdfs:subClassOf>
</owl:Class>
<owl:Class rdf:about="#Agent">
    <rdfs:subClassOf>
        <owl:Restriction>
            <owl:onProperty rdf:resource="#hasAttribute" />
            <owl:allValuesFrom rdf:resource="#Attribute" />
        </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
        <owl:Restriction>
            <owl:onProperty rdf:resource="#hasQuantity" />
            <owl:allValuesFrom rdf:resource="#Quantity" />
        </owl:Restriction>
    </rdfs:subClassOf>
</owl:Class>
<owl:ObjectProperty rdf:ID="hasConfiguration" />
<owl:ObjectProperty rdf:ID="hasAttribute" />
<owl:ObjectProperty rdf:ID="hasQuantity" />
```

B.4 Configuration Restrictions

```
<owl:Class rdf:about="#Configuration">
    <rdfs:subClassOf>
        <owl:Restriction>
            <owl:onProperty rdf:resource="#hasConfigurationTarget" />
```

```
            <owl:cardinality rdf:datatype="&xsd;nonNegativeInteger">1
            </owl:cardinality>
        </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
        <owl:Restriction>
            <owl:onProperty rdf:resource="#hasConfigurationOwner" />
            <owl:cardinality rdf:datatype="&xsd;nonNegativeInteger">1
            </owl:cardinality>
        </owl:Restriction>
    </rdfs:subClassOf>
</owl:Class>
<owl:ObjectProperty rdf:ID="hasConfigurationTarget" />
<owl:ObjectProperty rdf:ID="isConfigurationTargetOf">
    <owl:inverseOf rdf:resource="#hasConfigurationTarget" />
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="hasConfigurationOwner">
    <owl:inverseOf rdf:resource="#hasConfiguration" />
</owl:ObjectProperty>
```

B.4.1 Entity and Agent Configuration Restrictions

```
<owl:Class rdf:about="#Agent">
    <rdfs:subClassOf>
        <owl:Restriction>
            <owl:onProperty rdf:resource="#hasConfiguration" />
            <owl:allValuesFrom>
                <owl:Class>
                    <owl:intersectionOf rdf:parseType="Collection">
                        <owl:Class rdf:about="#Configuration" />
                        <owl:Restriction>
                            <owl:onProperty rdf:resource="#hasConfigurationTarget" />
                            <owl:someValuesFrom>
                                <owl:Class>
                                    <owl:unionOf rdf:parseType="Collection">
                                        <owl:Class rdf:about="#Entity" />
                                        <owl:Class rdf:about="#Agent" />
                                    </owl:unionOf>
                                </owl:Class>
                            </owl:someValuesFrom>
                        </owl:Restriction>
                    </owl:intersectionOf>
                </owl:Class>
            </owl:allValuesFrom>
        </owl:Restriction>
    </rdfs:subClassOf>
</owl:Class>
<owl:Class rdf:about="#Entity">
    <rdfs:subClassOf>
        <owl:Restriction>
            <owl:onProperty rdf:resource="#hasConfiguration" />
            <owl:allValuesFrom>
                <owl:Class>
                    <owl:intersectionOf rdf:parseType="Collection">
                        <owl:Class rdf:about="#Configuration" />
                        <owl:Restriction>
                            <owl:onProperty rdf:resource="#hasConfigurationTarget" />
                            <owl:someValuesFrom>
                                <owl:Class>
                                    <owl:unionOf rdf:parseType="Collection">
                                        <owl:Class rdf:about="#Entity" />
                                        <owl:Class rdf:about="#Agent" />
                                    </owl:unionOf>
```

```
                                </owl:Class>
                            </owl:someValuesFrom>
                        </owl:Restriction>
                    </owl:intersectionOf>
                </owl:Class>
            </owl:allValuesFrom>
        </owl:Restriction>
    </rdfs:subClassOf>
</owl:Class>
```

B.5 Attribute Restrictions

```
<owl:Class rdf:about="#Attribute">
    <rdfs:subClassOf>
        <owl:Restriction>
            <owl:onProperty rdf:resource="#hasAttributeValue" />
            <owl:allValuesFrom rdf:resource="#AttributeValue" />
        </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
        <owl:Restriction>
            <owl:onProperty rdf:resource="#hasAttributeValue" />
            <owl:cardinality rdf:datatype="&xsd;nonNegativeInteger">1
            </owl:cardinality>
        </owl:Restriction>
    </rdfs:subClassOf>
</owl:Class>
<owl:ObjectProperty rdf:ID="hasAttributeValue" />
```

B.6 Quantity Restrictions

```
<owl:Class rdf:about="#Quantity">
    <rdfs:subClassOf>
        <owl:Restriction>
            <owl:onProperty rdf:resource="#hasMagnitude" />
            <owl:cardinality rdf:datatype="&xsd;nonNegativeInteger">1
            </owl:cardinality>
        </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
        <owl:Restriction>
            <owl:onProperty rdf:resource="#hasMagnitude" />
            <owl:allValuesFrom rdf:resource="#Magnitude" />
        </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
        <owl:Restriction>
            <owl:onProperty rdf:resource="#hasDerivative" />
            <owl:cardinality rdf:datatype="&xsd;nonNegativeInteger">1
            </owl:cardinality>
        </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
        <owl:Restriction>
            <owl:onProperty rdf:resource="#hasDerivative" />
            <owl:allValuesFrom rdf:resource="#Derivative" />
        </owl:Restriction>
    </rdfs:subClassOf>
</owl:Class>
<owl:Class rdf:about="#Magnitude">
    <rdfs:subClassOf>
        <owl:Restriction>
            <owl:onProperty rdf:resource="#hasQuantitySpace" />
```

```
            <owl:cardinality rdf:datatype="&xsd;nonNegativeInteger">1
            </owl:cardinality>
        </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
        <owl:Restriction>
            <owl:onProperty rdf:resource="#hasQuantitySpace" />
            <owl:allValuesFrom rdf:resource="#QuantitySpace" />
        </owl:Restriction>
    </rdfs:subClassOf>
</owl:Class>
<owl:Class rdf:about="#Derivative">
    <rdfs:subClassOf>
        <owl:Restriction>
            <owl:onProperty rdf:resource="#hasQuantitySpace" />
            <owl:cardinality rdf:datatype="&xsd;nonNegativeInteger">1
            </owl:cardinality>
        </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
        <owl:Restriction>
            <owl:onProperty rdf:resource="#hasQuantitySpace" />
            <owl:allValuesFrom rdf:resource="#QuantitySpace" />
        </owl:Restriction>
    </rdfs:subClassOf>
</owl:Class>
<owl:ObjectProperty rdf:ID="hasMagnitude">
    <rdfs:range  rdf:resource="#Magnitude" />
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="magnitudeBelongsToQuantity">
    <owl:inverseOf rdf:resource="#hasMagnitude" />
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="hasDerivative">
    <rdfs:range  rdf:resource="#Derivative" />
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="derivativeBelongsToQuantity">
    <owl:inverseOf rdf:resource="#hasDerivative" />
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="hasQuantitySpace">
    <rdfs:range  rdf:resource="#QuantitySpace" />
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="isQuantitySpaceOf">
    <owl:inverseOf rdf:resource="#hasQuantitySpace" />
    <rdfs:domain rdf:resource="#QuantitySpace" />
</owl:ObjectProperty>
```

B.7 Quantity Space Restrictions

```
<owl:Class rdf:about="#QuantitySpace">
    <rdfs:subClassOf>
        <owl:Restriction>
            <owl:onProperty rdf:resource="#containsQualitativeValue" />
            <owl:minCardinality rdf:datatype="&xsd;nonNegativeInteger">1
            </owl:minCardinality>
        </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
        <owl:Restriction>
            <owl:onProperty rdf:resource="#containsQualitativeValue" />
            <owl:allValuesFrom rdf:resource="#QualitativeValue" />
        </owl:Restriction>
    </rdfs:subClassOf>
</owl:Class>
```

```
<owl:ObjectProperty rdf:ID="containsQualitativeValue">
    <rdfs:range  rdf:resource="#QualitativeValue" />
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="belongsToQuantitySpace">
    <owl:inverseOf rdf:resource="#containsQualitativeValue" />
</owl:ObjectProperty>
```

B.8 Causal Dependency Restrictions

B.8.1 Quantity Causal Dependency Restrictions

```
<owl:Class rdf:about="#Quantity">
    <rdfs:subClassOf>
        <owl:Restriction>
            <owl:onProperty rdf:resource="#hasCausalDependency" />
            <owl:allValuesFrom>
                <owl:Class>
                    <owl:intersectionOf rdf:parseType="Collection">
                        <owl:Class>
                            <owl:unionOf rdf:parseType="Collection">
                                <owl:Class rdf:about="#Influence" />
                                <owl:Class rdf:about="#Proportionality" />
                            </owl:unionOf>
                        </owl:Class>
                        <owl:Restriction>
                            <owl:onProperty rdf:resource="#hasCausalDependencyTarget" />
                            <owl:allValuesFrom rdf:resource="#Quantity" />
                        </owl:Restriction>
                        <owl:Restriction>
                            <owl:onProperty rdf:resource="#hasCausalDependencyTarget" />
                            <owl:cardinality rdf:datatype="&xsd;nonNegativeInteger">1
                            </owl:cardinality>
                        </owl:Restriction>
                    </owl:intersectionOf>
                </owl:Class>
            </owl:allValuesFrom>
        </owl:Restriction>
    </rdfs:subClassOf>
</owl:Class>
<owl:ObjectProperty rdf:ID="hasCausalDependency" />
<owl:ObjectProperty rdf:ID="hasCausalDependencyTarget" />
```

B.9 Correspondences

```
<owl:ObjectProperty rdf:ID="hasCorrespondence" />
<owl:ObjectProperty rdf:ID="hasCorrespondenceTarget" />
<owl:Class rdf:about="#QuantitySpace">
    <rdfs:subClassOf>
        <owl:Restriction>
            <owl:onProperty rdf:resource="#hasCorrespondence" />
            <owl:allValuesFrom>
                <owl:Class>
                    <owl:intersectionOf rdf:parseType="Collection">
                        <owl:Class rdf:about="#QuantitySpaceCorrespondence" />
                        <owl:Restriction>
                            <owl:onProperty rdf:resource="#hasCorrespondenceTarget" />
                            <owl:allValuesFrom rdf:resource="#QuantitySpace" />
                        </owl:Restriction>
                        <owl:Restriction>
                            <owl:onProperty rdf:resource="#hasCorrespondenceTarget" />
                            <owl:cardinality rdf:datatype="&xsd;nonNegativeInteger">1
                            </owl:cardinality>
                        </owl:Restriction>
```

```
                    </owl:intersectionOf>
                </owl:Class>
            </owl:allValuesFrom>
            <!-- The domain and range should have the same cardinality, but this
            is not expressable in OWL -->
        </owl:Restriction>
    </rdfs:subClassOf>
</owl:Class>
<owl:Class rdf:about="#QualitativeValue">
    <rdfs:subClassOf>
        <owl:Restriction>
            <owl:onProperty rdf:resource="#hasCorrespondence" />
            <owl:allValuesFrom>
                <owl:Class>
                    <owl:intersectionOf rdf:parseType="Collection">
                        <owl:Class rdf:about="#ValueCorrespondence" />
                        <owl:Restriction>
                            <owl:onProperty rdf:resource="#hasCorrespondenceTarget" />
                            <owl:allValuesFrom rdf:resource="#QualitativeValue" />
                        </owl:Restriction>
                        <owl:Restriction>
                            <owl:onProperty rdf:resource="#hasCorrespondenceTarget" />
                            <owl:cardinality rdf:datatype="&xsd;nonNegativeInteger">1
                            </owl:cardinality>
                        </owl:Restriction>
                    </owl:intersectionOf>
                </owl:Class>
            </owl:allValuesFrom>
            <!-- The qualitative values should not be in the same
            quantity space, but this is not expressable in OWL -->
        </owl:Restriction>
    </rdfs:subClassOf>
</owl:Class>
```

B.10 Inequalities

B.10.1 Inequalities Belonging to Points

```
<owl:Class rdf:ID="PointBelongingToMagnitude">
    <owl:equivalentClass>
        <owl:Class>
            <owl:intersectionOf rdf:parseType="Collection">
                <owl:Class rdf:about="#Point" />
                <owl:Restriction>
                    <owl:onProperty rdf:resource="#belongsToQuantitySpace" />
                    <owl:someValuesFrom>
                        <owl:Class>
                            <owl:intersectionOf rdf:parseType="Collection">
                                <owl:Class rdf:about="#QuantitySpace" />
                                <owl:Restriction>
                                    <owl:onProperty rdf:resource="#isQuantitySpaceOf" />
                                    <owl:someValuesFrom rdf:resource="#Magnitude" />
                                </owl:Restriction>
                            </owl:intersectionOf>
                        </owl:Class>
                    </owl:someValuesFrom>
                </owl:Restriction>
            </owl:intersectionOf>
        </owl:Class>
    </owl:equivalentClass>
    <rdfs:subClassOf>
        <owl:Restriction>
            <owl:onProperty rdf:resource="#hasInequality" />
```

```
            <owl:allValuesFrom rdf:resource="#PointBelongingToMagnitude" />
        </owl:Restriction>
    </rdfs:subClassOf>
    <owl:disjointWith rdf:resource="#PointBelongingToDerivative" />
</owl:Class>
<owl:Class rdf:ID="PointBelongingToDerivative">
    <owl:equivalentClass>
        <owl:Class>
            <owl:intersectionOf rdf:parseType="Collection">
                <owl:Class rdf:about="#Point" />
                <owl:Restriction>
                    <owl:onProperty rdf:resource="#belongsToQuantitySpace" />
                    <owl:someValuesFrom>
                        <owl:Class>
                            <owl:intersectionOf rdf:parseType="Collection">
                                <owl:Class rdf:about="#QuantitySpace" />
                                <owl:Restriction>
                                    <owl:onProperty rdf:resource="#isQuantitySpaceOf" />
                                    <owl:someValuesFrom rdf:resource="#Derivative" />
                                </owl:Restriction>
                            </owl:intersectionOf>
                        </owl:Class>
                    </owl:someValuesFrom>
                </owl:Restriction>
            </owl:intersectionOf>
        </owl:Class>
    </owl:equivalentClass>
    <rdfs:subClassOf>
        <owl:Restriction>
            <owl:onProperty rdf:resource="#hasInequality" />
            <owl:allValuesFrom rdf:resource="#PointBelongingToMagnitude" />
        </owl:Restriction>
    </rdfs:subClassOf>
    <owl:disjointWith rdf:resource="#PointBelongingToMagnitude" />
</owl:Class>
```

B.10.2 Inequalities Belonging to Magnitudes or Derivatives

```
<owl:ObjectProperty rdf:ID="hasInequalityTarget">
    <rdfs:domain rdf:resource="#Inequality" />
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="hasInequality">
    <rdfs:range rdf:resource="#Inequality" />
</owl:ObjectProperty>
<!-- Magnitudes can have inequalities with other magnitudes or
points (qualitative values) within their own quantity space as
targets -->
<owl:Class rdf:about="#Magnitude">
    <rdfs:subClassOf>
        <owl:Restriction>
            <owl:onProperty rdf:resource="#hasInequality" />
            <owl:allValuesFrom>
                <owl:Class>
                    <owl:intersectionOf rdf:parseType="Collection">
                        <owl:Class rdf:about="#Inequality" />
                        <owl:Restriction>
                            <owl:onProperty rdf:resource="#hasInequalityTarget" />
                            <owl:allValuesFrom>
                                <owl:Class>
                                    <owl:unionOf rdf:parseType="Collection">
                                        <owl:Class rdf:about="#Magnitude" />
                                        <!-- The magnitude should not be the same individual as
                                        the domain. -->
```

```
                                    <owl:Class rdf:about="#Point" />
                                    <!-- The point should be in the same quantity space,
                                    but this is not expressable in OWL -->
                                </owl:unionOf>
                            </owl:Class>
                        </owl:allValuesFrom>
                    </owl:Restriction>
                </owl:intersectionOf>
            </owl:Class>
        </owl:allValuesFrom>
    </owl:Restriction>
</rdfs:subClassOf>
</owl:Class>
<!-- Derivatives can have inequalities with other derivatives or
points (qualitative values) within their own quantity space as
targets -->
<owl:Class rdf:about="#Derivative">
    <rdfs:subClassOf>
        <owl:Restriction>
            <owl:onProperty rdf:resource="#hasInequality" />
            <owl:allValuesFrom>
                <owl:Class>
                    <owl:intersectionOf rdf:parseType="Collection">
                        <owl:Class rdf:about="#Inequality" />
                        <owl:Restriction>
                            <owl:onProperty rdf:resource="#hasInequalityTarget" />
                            <owl:allValuesFrom>
                                <owl:Class>
                                    <owl:unionOf rdf:parseType="Collection">
                                        <owl:Class rdf:about="#Derivative" />
                                        <!-- The derivative should not be the same individual
                                        as the domain, but that is not expressable in OWL. -->
                                        <owl:Class rdf:about="#Point" />
                                        <!-- The point should be in the quantity space of the
                                        derivative, but this is not expressable in OWL -->
                                    </owl:unionOf>
                                </owl:Class>
                            </owl:allValuesFrom>
                        </owl:Restriction>
                    </owl:intersectionOf>
                </owl:Class>
            </owl:allValuesFrom>
        </owl:Restriction>
    </rdfs:subClassOf>
</owl:Class>
```

B.10.3 Inequalities belonging to Intervals

```
<owl:Class rdf:about="#Interval">
    <rdfs:subClassOf>
        <owl:Restriction>
            <owl:onProperty rdf:resource="#hasInequality" />
            <owl:allValuesFrom>
                <owl:Class>
                    <owl:intersectionOf rdf:parseType="Collection">
                        <owl:Class rdf:about="#Inequality" />
                        <owl:Restriction>
                            <owl:onProperty rdf:resource="#hasInequalityTarget" />
                            <owl:minCardinality rdf:datatype="&xsd;nonNegativeInteger">1
                            </owl:minCardinality>
                        </owl:Restriction>
                        <owl:Restriction>
                            <owl:onProperty rdf:resource="#hasInequalityTarget" />
```

```
                        <owl:allValuesFrom rdf:resource="#Point" />
                    </owl:Restriction>
                </owl:intersectionOf>
            </owl:Class>
        </owl:allValuesFrom>
    </owl:Restriction>
</rdfs:subClassOf>
</owl:Class>
```

B.10.4 hasValue Relations

```
<owl:ObjectProperty rdf:ID="hasValueTarget" />
<owl:Class rdf:about="#hasValue">
    <rdfs:subClassOf>
        <owl:Restriction>
            <owl:onProperty rdf:resource="#hasValueTarget" />
            <owl:allValuesFrom rdf:resource="#QualitativeValue" />
        </owl:Restriction>
    </rdfs:subClassOf>
</owl:Class>
```

B.11 PlusMin Relations

```
<owl:ObjectProperty rdf:ID="hasLefthandSide">
    <owl:inverseOf rdf:resource="#isLefthandSideOf" />
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="isLefthandSideOf">
    <owl:inverseOf rdf:resource="#hasLefthandSide" />
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="hasRighthandSide" />
<owl:Class rdf:about="#Derivative">
    <rdfs:subClassOf>
        <owl:Restriction>
            <owl:onProperty rdf:resource="#isLefthandSideOf" />
            <owl:allValuesFrom>
                <owl:Class>
                    <owl:intersectionOf rdf:parseType="Collection">
                        <owl:Class rdf:about="#PlusMin" />
                        <owl:Restriction>
                            <owl:onProperty rdf:resource="#hasLefthandSide" />
                            <owl:cardinality rdf:datatype="&xsd;nonNegativeInteger">1
                            </owl:cardinality>
                        </owl:Restriction>
                        <owl:Restriction>
                            <owl:onProperty rdf:resource="#hasRighthandSide" />
                            <owl:allValuesFrom rdf:resource="#Derivative" />
                        </owl:Restriction>
                        <owl:Restriction>
                            <owl:onProperty rdf:resource="#hasRighthandSide" />
                            <owl:cardinality rdf:datatype="&xsd;nonNegativeInteger">1
                            </owl:cardinality>
                        </owl:Restriction>
                        <owl:Restriction>
                            <owl:onProperty rdf:resource="#hasInequality" />
                            <owl:allValuesFrom>
                                <owl:Class>
                                    <owl:intersectionOf rdf:parseType="Collection">
                                        <owl:Class rdf:about="#Inequality" />
                                        <owl:Restriction>
                                            <owl:onProperty rdf:resource="#hasInequalityTarget" />
                                            <owl:allValuesFrom rdf:resource="#Derivative" />
                                        </owl:Restriction>
                                    </owl:intersectionOf>
```

```
                            </owl:Class>
                        </owl:allValuesFrom>
                    </owl:Restriction>
                </owl:intersectionOf>
            </owl:Class>
        </owl:allValuesFrom>
    </owl:Restriction>
</rdfs:subClassOf>
</owl:Class>
<owl:Class rdf:about="#Magnitude">
    <rdfs:subClassOf>
        <owl:Restriction>
            <owl:onProperty rdf:resource="#isLefthandSideOf" />
            <owl:allValuesFrom>
                <owl:Class>
                    <owl:intersectionOf rdf:parseType="Collection">
                        <owl:Class rdf:about="#PlusMin" />
                        <owl:Restriction>
                            <owl:onProperty rdf:resource="#hasLefthandSide" />
                            <owl:cardinality rdf:datatype="&xsd;nonNegativeInteger">1
                            </owl:cardinality>
                        </owl:Restriction>
                        <owl:Restriction>
                            <owl:onProperty rdf:resource="#hasRighthandSide" />
                            <owl:allValuesFrom>
                                <owl:Class>
                                    <owl:unionOf rdf:parseType="Collection">
                                        <owl:Class rdf:about="#Magnitude" />
                                        <owl:Class rdf:about="#PointBelongingToMagnitude" />
                                    </owl:unionOf>
                                </owl:Class>
                            </owl:allValuesFrom>
                        </owl:Restriction>
                        <owl:Restriction>
                            <owl:onProperty rdf:resource="#hasRighthandSide" />
                            <owl:cardinality rdf:datatype="&xsd;nonNegativeInteger">1
                            </owl:cardinality>
                        </owl:Restriction>
                        <owl:Restriction>
                            <owl:onProperty rdf:resource="#hasInequality" />
                            <owl:allValuesFrom>
                                <owl:Class>
                                    <owl:intersectionOf rdf:parseType="Collection">
                                        <owl:Class rdf:about="#Inequality" />
                                        <owl:Restriction>
                                            <owl:onProperty rdf:resource="#hasInequalityTarget" />
                                            <owl:allValuesFrom>
                                                <owl:Class>
                                                    <owl:unionOf rdf:parseType="Collection">
                                                        <owl:Class rdf:about="#Magnitude" />
                                                        <owl:Class rdf:about="#PointBelongingToMagnitude" />
                                                    </owl:unionOf>
                                                </owl:Class>
                                            </owl:allValuesFrom>
                                        </owl:Restriction>
                                    </owl:intersectionOf>
                                </owl:Class>
                            </owl:allValuesFrom>
                        </owl:Restriction>
                    </owl:intersectionOf>
                </owl:Class>
            </owl:allValuesFrom>
        </owl:Restriction>
```

```
    </rdfs:subClassOf>
</owl:Class>
<owl:Class rdf:about="#Point">
    <rdfs:subClassOf>
        <owl:Restriction>
            <owl:onProperty rdf:resource="#isLefthandSideOf" />
            <owl:allValuesFrom>
                <owl:Class>
                    <owl:intersectionOf rdf:parseType="Collection">
                        <owl:Class rdf:about="#PlusMin" />
                        <owl:Restriction>
                            <owl:onProperty rdf:resource="#hasLefthandSide" />
                            <owl:cardinality rdf:datatype="&xsd;nonNegativeInteger">1
                            </owl:cardinality>
                        </owl:Restriction>
                        <owl:Restriction>
                            <owl:onProperty rdf:resource="#hasRighthandSide" />
                            <owl:allValuesFrom>
                                <owl:Class>
                                    <owl:unionOf rdf:parseType="Collection">
                                        <owl:Class rdf:about="#Magnitude" />
                                        <owl:Class rdf:about="#PointBelongingToMagnitude" />
                                    </owl:unionOf>
                                </owl:Class>
                            </owl:allValuesFrom>
                        </owl:Restriction>
                        <owl:Restriction>
                            <owl:onProperty rdf:resource="#hasRighthandSide" />
                            <owl:cardinality rdf:datatype="&xsd;nonNegativeInteger">1
                            </owl:cardinality>
                        </owl:Restriction>
                        <owl:Restriction>
                            <owl:onProperty rdf:resource="#hasInequality" />
                            <owl:allValuesFrom>
                                <owl:Class>
                                    <owl:intersectionOf rdf:parseType="Collection">
                                        <owl:Class rdf:about="#Inequality" />
                                        <owl:Restriction>
                                            <owl:onProperty rdf:resource="#hasInequalityTarget" />
                                            <owl:allValuesFrom>
                                                <owl:Class>
                                                    <owl:unionOf rdf:parseType="Collection">
                                                        <owl:Class rdf:about="#Magnitude" />
                                                        <owl:Class rdf:about="#PointBelongingToMagnitude" />
                                                    </owl:unionOf>
                                                </owl:Class>
                                            </owl:allValuesFrom>
                                        </owl:Restriction>
                                    </owl:intersectionOf>
                                </owl:Class>
                            </owl:allValuesFrom>
                        </owl:Restriction>
                    </owl:intersectionOf>
                </owl:Class>
            </owl:allValuesFrom>
        </owl:Restriction>
    </rdfs:subClassOf>
</owl:Class>
```

B.12 Aggregate

B.12.1 Model Fragment

```
<owl:Class rdf:about="#ModelFragment">
    <rdfs:subClassOf>
        <owl:Restriction>
            <owl:onProperty rdf:resource="#hasCondition" />
            <owl:allValuesFrom>
                <owl:Class>
                    <owl:unionOf rdf:parseType="Collection">
                        <owl:Class rdf:about="#Structural" />
                        <owl:Class rdf:about="#Behavioural" />
                        <owl:Class rdf:about="#AssumptionType" />
                        <owl:Class rdf:about="#Inequality" />
                        <owl:Class rdf:about="#PlusMin" />
                        <owl:Class rdf:about="#ModelFragment" />
                    </owl:unionOf>
                </owl:Class>
            </owl:allValuesFrom>
        </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
        <owl:Restriction>
            <owl:onProperty rdf:resource="#hasConsequence" />
            <owl:allValuesFrom>
                <owl:Class>
                    <owl:unionOf rdf:parseType="Collection">
                        <owl:Class rdf:about="#Entity" />
                        <owl:Class rdf:about="#Configuration" />
                        <owl:Class rdf:about="#Attribute" />
                        <owl:Class rdf:about="#AttributeValue" />
                        <owl:Class rdf:about="#Behavioural" />
                        <owl:Class rdf:about="#Dependency" />
                    </owl:unionOf>
                </owl:Class>
            </owl:allValuesFrom>
        </owl:Restriction>
    </rdfs:subClassOf>
</owl:Class>
<owl:ObjectProperty rdf:ID="hasCondition" />
<owl:ObjectProperty rdf:ID="hasConsequence" />
```

B.12.2 Static Fragment

```
<owl:Class rdf:about="#StaticFragment">
    <owl:equivalentClass>
        <owl:Class>
            <owl:intersectionOf rdf:parseType="Collection">
                <owl:Class rdf:about="#ModelFragment" />
                <owl:Class>
                    <owl:complementOf>
                        <owl:Restriction>
                            <owl:onProperty rdf:resource="#hasConsequence" />
                            <owl:someValuesFrom rdf:resource="#Influence" />
                        </owl:Restriction>
                    </owl:complementOf>
                </owl:Class>
                <owl:Class>
                    <owl:complementOf>
                        <owl:Restriction>
                            <owl:onProperty rdf:resource="#hasCondition" />
                            <owl:someValuesFrom rdf:resource="#Agent" />
                        </owl:Restriction>
```

```
                </owl:complementOf>
            </owl:Class>
            <owl:Class>
                <owl:complementOf>
                    <owl:Restriction>
                        <owl:onProperty rdf:resource="#hasConsequence" />
                        <owl:someValuesFrom rdf:resource="#Entity" />
                    </owl:Restriction>
                </owl:complementOf>
            </owl:Class>
            <owl:Class>
                <owl:complementOf>
                    <owl:Restriction>
                        <owl:onProperty rdf:resource="#hasConsequence" />
                        <owl:someValuesFrom rdf:resource="#Configuration" />
                    </owl:Restriction>
                </owl:complementOf>
            </owl:Class>
        </owl:intersectionOf>
    </owl:Class>
</owl:equivalentClass>
</owl:Class>
```

B.12.3 Process Fragment

```
<owl:Class rdf:about="#ProcessFragment">
    <owl:equivalentClass>
        <owl:Class>
            <owl:intersectionOf rdf:parseType="Collection">
                <owl:Class rdf:about="#ModelFragment" />
                <owl:Restriction>
                    <owl:onProperty rdf:resource="#hasConsequence" />
                    <owl:someValuesFrom rdf:resource="#Influence" />
                </owl:Restriction>
                <owl:Class>
                    <owl:complementOf>
                        <owl:Restriction>
                            <owl:onProperty rdf:resource="#hasCondition" />
                            <owl:someValuesFrom rdf:resource="#Agent" />
                        </owl:Restriction>
                    </owl:complementOf>
                </owl:Class>
            </owl:intersectionOf>
        </owl:Class>
    </owl:equivalentClass>
</owl:Class>
```

B.12.4 Agent Fragment

```
<owl:Class rdf:about="#AgentFragment">
    <owl:equivalentClass>
        <owl:Class>
            <owl:intersectionOf rdf:parseType="Collection">
                <owl:Class rdf:about="#ModelFragment" />
                <owl:Restriction>
                    <owl:onProperty rdf:resource="#hasCondition" />
                    <owl:someValuesFrom rdf:resource="#Agent" />
                </owl:Restriction>
            </owl:intersectionOf>
        </owl:Class>
    </owl:equivalentClass>
</owl:Class>
```

B.12.5 Scenario

```
<owl:Class rdf:about="#Scenario">
    <rdfs:subClassOf>
        <owl:Restriction>
            <owl:onProperty rdf:resource="#hasFact" />
            <owl:allValuesFrom>
                <owl:Class>
                    <owl:unionOf rdf:parseType="Collection">
                        <owl:Class rdf:about="#Structural" />
                        <owl:Class rdf:about="#Behavioural" />
                        <owl:Class rdf:about="#AssumptionType" />
                        <owl:Class rdf:about="#Inequality" />
                        <owl:Class rdf:about="#PlusMin" />
                    </owl:unionOf>
                </owl:Class>
            </owl:allValuesFrom>
        </owl:Restriction>
    </rdfs:subClassOf>
</owl:Class>
<owl:ObjectProperty rdf:ID="hasFact" />
```

C Example Qualitative Model Definitions

C.1 Entity and Agent Definitions

```
<owl:Class rdf:about="&qrm;Object"
    rdfs:comment=""
    rdfs:label="Object">
  <rdfs:subClassOf rdf:resource="&qr;Entity"/>
  <owl:disjointWith rdf:resource="&qrm;Substance"/>
</owl:Class>

<owl:Class rdf:about="&qrm;Container"
    rdfs:comment=""
    rdfs:label="Container">
  <rdfs:subClassOf rdf:resource="&qrm;Object"/>
  <owl:disjointWith rdf:resource="&qrm;Tube"/>
</owl:Class>

<owl:Class rdf:about="&qrm;Tube"
    rdfs:comment=""
    rdfs:label="Tube">
  <rdfs:subClassOf rdf:resource="&qrm;Object"/>
  <owl:disjointWith rdf:resource="&qrm;Container"/>
</owl:Class>

<owl:Class rdf:about="&qrm;Substance"
    rdfs:comment=""
    rdfs:label="Substance">
  <rdfs:subClassOf rdf:resource="&qr;Entity"/>
  <owl:disjointWith rdf:resource="&qrm;Object"/>
</owl:Class>
```

C.2 Configuration Definitions

```
<owl:Class rdf:about="&qrm;Contains"
    rdfs:comment=""
    rdfs:label="Contains">
  <rdfs:subClassOf rdf:resource="&qr;Configuration"/>
  <owl:disjointWith rdf:resource="&qrm;Connected"/>
</owl:Class>
```

```
<owl:Class rdf:about="&qrm;Connected"
    rdfs:comment="Om aan te geven dat twee dingen aan elkaar zitten. :P"
    rdfs:label="Connected">
  <rdfs:subClassOf rdf:resource="&qr;Configuration"/>
  <owl:disjointWith rdf:resource="&qrm;Contains"/>
</owl:Class>
```

C.3 Assumptions

```
<owl:Class rdf:about="&qrm;Containersnotempty"
    rdfs:comment="Assume the containers are not empty"
    rdfs:label="Containersnotempty">
    <rdfs:subClassOf rdf:resource="&qr;Assumption"/>
</owl:Class>
```

C.4 Attribute Definition

```
<owl:Class rdf:about="&qrm;Openorclosed"
    rdfs:comment="Indicates if something is open or closed."
    rdfs:label="Openorclosed">
    <rdfs:subClassOf>
        <owl:Restriction>
            <owl:allValuesFrom rdf:resource="&qrm;OpenorclosedValue"/>
            <owl:onProperty rdf:resource="&qr;hasAttributeValue"/>
        </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf rdf:resource="&qr;Attribute"/>
    <owl:disjointWith rdf:resource="&qrm;Wakkerofslapen"/>
</owl:Class>
<owl:Class rdf:about="&qrm;OpenorclosedValue" rdfs:label="OpenorclosedValue">
    <rdfs:subClassOf rdf:resource="&qr;AttributeValue"/>
    <owl:disjointWith rdf:resource="&qrm;WakkerofslapenValue"/>
    <owl:oneOf rdf:parseType="Collection">
        <owl:Thing rdf:about="http://phyro.mine.nu/model.owl#Open"/>
        <owl:Thing rdf:about="http://phyro.mine.nu/model.owl#Closed"/>
    </owl:oneOf>
</owl:Class>
<qrm:OpenorclosedValue rdf:about="&qrm;Open"
    rdfs:label="Open"/>
<qrm:OpenorclosedValue rdf:about="&qrm;Closed"
    rdfs:label="Closed"/>
<owl:AllDifferent>
    <owl:distinctMembers rdf:parseType="Collection">
        <owl:Thing rdf:about="http://phyro.mine.nu/model.owl#Open"/>
        <owl:Thing rdf:about="http://phyro.mine.nu/model.owl#Closed"/>
    </owl:distinctMembers>
</owl:AllDifferent>
```

C.5 Quantity Space Definitions

```
<owl:Class rdf:about="&qrm;Negativezeropositive"
    rdfs:comment=""
    rdfs:label="Negativezeropositive">
    <rdfs:subClassOf>
        <owl:Restriction>
            <owl:onProperty rdf:resource="&qr;containsQualitativeValue"/>
            <owl:someValuesFrom>
                <owl:Class>
                    <owl:intersectionOf rdf:parseType="Collection">
                        <owl:Class rdf:about="http://phyro.mine.nu/model.owl#Negative"/>
                        <owl:Restriction>
                            <owl:onProperty rdf:resource="&qr;hasInequality"/>
```

```
                        <owl:someValuesFrom>
                            <owl:Class>
                                <owl:intersectionOf rdf:parseType="Collection">
                                    <owl:Class rdf:about="http://phyro.mine.nu/qrontology.owl#SmallerThan"/>
                                    <owl:Restriction>
                                        <owl:hasValue rdf:resource="&qr;Zero"/>
                                        <owl:onProperty rdf:resource="&qr;hasInequalityTarget"/>
                                    </owl:Restriction>
                                </owl:intersectionOf>
                            </owl:Class>
                        </owl:someValuesFrom>
                    </owl:Restriction>
                </owl:intersectionOf>
            </owl:Class>
        </owl:someValuesFrom>
    </owl:Restriction>
</rdfs:subClassOf>
<rdfs:subClassOf>
    <owl:Restriction>
        <owl:hasValue rdf:resource="&qr;Zero"/>
        <owl:onProperty rdf:resource="&qr;containsQualitativeValue"/>
    </owl:Restriction>
</rdfs:subClassOf>
<rdfs:subClassOf>
    <owl:Restriction>
        <owl:onProperty rdf:resource="&qr;containsQualitativeValue"/>
        <owl:someValuesFrom>
            <owl:Class>
                <owl:intersectionOf rdf:parseType="Collection">
                    <owl:Class rdf:about="http://phyro.mine.nu/model.owl#Positive"/>
                    <owl:Restriction>
                        <owl:onProperty rdf:resource="&qr;hasInequality"/>
                        <owl:someValuesFrom>
                            <owl:Class>
                                <owl:intersectionOf rdf:parseType="Collection">
                                    <owl:Class rdf:about="http://phyro.mine.nu/qrontology.owl#GreaterThan"/>
                                    <owl:Restriction>
                                        <owl:hasValue rdf:resource="&qr;Zero"/>
                                        <owl:onProperty rdf:resource="&qr;hasInequalityTarget"/>
                                    </owl:Restriction>
                                </owl:intersectionOf>
                            </owl:Class>
                        </owl:someValuesFrom>
                    </owl:Restriction>
                </owl:intersectionOf>
            </owl:Class>
        </owl:someValuesFrom>
    </owl:Restriction>
</rdfs:subClassOf>
<rdfs:subClassOf rdf:resource="&qr;QuantitySpace"/>
<owl:disjointWith rdf:resource="&qrm;Minimumnegativezeropositivemaximum"/>
<owl:disjointWith rdf:resource="&qrm;Mzp"/>
<owl:disjointWith rdf:resource="&qrm;Zeropositivemaximum"/>
</owl:Class>
<owl:Class rdf:about="&qrm;Negative"
    rdfs:label="Negative">
    <owl:disjointWith rdf:resource="&qrm;Maximum"/>
    <owl:disjointWith rdf:resource="&qrm;Min"/>
    <owl:disjointWith rdf:resource="&qrm;Minimum"/>
    <owl:disjointWith rdf:resource="&qrm;Plus"/>
    <owl:disjointWith rdf:resource="&qrm;Positive"/>
    <rdfs:subClassOf rdf:resource="&qr;Interval"/>
</owl:Class>
```

```
<owl:Class rdf:about="&qrm;Positive"
    rdfs:label="Positive">
    <owl:disjointWith rdf:resource="&qrm;Maximum"/>
    <owl:disjointWith rdf:resource="&qrm;Min"/>
    <owl:disjointWith rdf:resource="&qrm;Minimum"/>
    <owl:disjointWith rdf:resource="&qrm;Negative"/>
    <owl:disjointWith rdf:resource="&qrm;Plus"/>
    <rdfs:subClassOf rdf:resource="&qr;Interval"/>
</owl:Class>
```

C.6 Quantity Definitions

```
<owl:Class rdf:about="&qrm;Flow"
    rdfs:comment=""
    rdfs:label="Flow">
    <rdfs:subClassOf>
        <owl:Restriction>
            <owl:onProperty rdf:resource="&qr;hasMagnitude"/>
            <owl:someValuesFrom>
                <owl:Class>
                    <owl:intersectionOf rdf:parseType="Collection">
                        <owl:Class rdf:about="http://phyro.mine.nu/qrontology.owl#Magnitude"/>
                        <owl:Restriction>
                            <owl:allValuesFrom>
                                <owl:Class>
                                    <owl:unionOf rdf:parseType="Collection">
                                <owl:Class rdf:about="&qrm;Minimumnegativezeropositivemaximum" />
                                <owl:Class rdf:about="&qrm;Negativezeropositive"/>
                                <owl:Class rdf:about="&qrm;Zeropositivemaximum"/>
                                    </owl:unionOf>
                                </owl:Class>
                            </owl:allValuesFrom>
                            <owl:onProperty rdf:resource="&qr;hasQuantitySpace"/>
                        </owl:Restriction>
                    </owl:intersectionOf>
                </owl:Class>
            </owl:someValuesFrom>
        </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf rdf:resource="&qr;Quantity"/>
</owl:Class>
```

C.7 Model Fragment Definitions

```
<qr:ModelFragment rdf:ID="WaterInContainer">
    <qr:hasCondition>
        <qrm:Container rdf:ID="Container1">
            <qr:hasConfiguration>
                <qrm:Contains>
                    <qr:hasConfigurationTarget rdf:resource="&qrm;Fluid1" />
                </qrm:Contains>
            </qr:hasConfiguration>
        </qrm:Container>
    </qr:hasCondition>
    <qr:hasCondition>
        <qrm:Fluid rdf:ID="Fluid1">
            <qr:hasQuantity rdf:resource="&qrm;Height1" />
        </qrm:Fluid>
    </qr:hasCondition>
    <qr:hasCondition>
        <qrm:Height rdf:ID="Height1">
            <qr:hasInequality>
                <qr:EqualTo>
```

```
        <qr:hasInequalityTarget rdf:resource="&qrm;Pressure1" />
    </qr:EqualTo>
</qr:hasInequality>
<qr:hasCausalDependency>
    <qr:PositiveProportionality>
        <qr:hasCausalDependencyTarget rdf:resource="&qrm;Pressure1" />
    </qr:PositiveProportionality>
</qr:hasCausalDependency>
<qr:hasMagnitude>
    <qr:Magnitude rdf:ID="MagnitudeHeight1">
        <qr:hasQuantitySpace>
            <qrm:Zeropositivemaximum rdf:ID="Zeropositivemaximum1">
                <qr:containsQualitativeValue rdf:resource="&qr;Zero" />
                <qr:containsQualitativeValue>
                    <qr:Positive rdf:ID="ZeropositivemaximumPositive1">
                        <qr:hasInequality>
                            <qr:SmallerThan>
                        <qr:hasInequalityTarget rdf:resource="&qrm;ZeropositivemaximumMaximum1" />
                            </qr:SmallerThan>
                        </qr:hasInequality>
                        <qr:hasInequality>
                            <qr:GreaterThan>
                                <qr:hasInequalityTarget rdf:resource="&qr;Zero" />
                            </qr:GreaterThan>
                        </qr:hasInequality>
                    </qr:Positive>
                </qr:containsQualitativeValue>
                <qr:containsQualitativeValue>
                    <qr:Maximum rdf:ID="ZeropositivemaximumMaximum1" />
                </qr:containsQualitativeValue>
                <qr:hasCorrespondence>
                    <qr:QuantityCorrespondence rdf:ID="Correspondence1">
                        <qr:hasCorrespondenceTarget rdf:resource="#Zeropositivemaximum2" />
                    </qr:QuantityCorrespondence>
                </qr:hasCorrespondence>
            </qrm:Zeropositivemaximum>
        </qr:hasQuantitySpace>
    </qr:Magnitude>
</qr:hasMagnitude>
<qr:hasDerivative>
    <qr:Derivative rdf:ID="DerivativeHeight1">
        <qr:hasQuantitySpace>
            <qrm:Mzp rdf:ID="Mzp1">
                <qr:containsQualitativeValue rdf:resource="&qr;Zero" />
                <qr:containsQualitativeValue>
                    <qr:Min rdf:ID="MzpMin1">
                        <qr:hasInequality>
                            <qr:SmallerThan>
                                <qr:hasInequalityTarget rdf:resource="&qr;Zero" />
                            </qr:SmallerThan>
                        </qr:hasInequality>
                    </qr:Min>
                </qr:containsQualitativeValue>
                <qr:containsQualitativeValue>
                    <qr:Plus rdf:ID="MzpPlus1">
                        <qr:hasInequality>
                            <qr:GreaterThan>
                                <qr:hasInequalityTarget rdf:resource="&qr;Zero" />
                            </qr:GreaterThan>
                        </qr:hasInequality>
                    </qr:Plus>
                </qr:containsQualitativeValue>
            </qrm:Mzp>
```

```
                </qr:hasQuantitySpace>
            </qr:Derivative>
        </qr:hasDerivative>
    </qrm:Height>
</qr:hasCondition>
<qr:hasConsequence>
    <qrm:Pressure rdf:ID="Pressure1">
        <qr:hasMagnitude>
            <qr:Magnitude rdf:ID="MagnitudePressure1">
                <qr:hasQuantitySpace>
                    <qrm:Zeropositivemaximum rdf:ID="Zeropositivemaximum2">
                        <qr:containsQualitativeValue>
                            <qr:Zero rdf:ID="ZeropositivemaximumZero2" />
                        </qr:containsQualitativeValue>
                        <qr:containsQualitativeValue>
                            <qr:Positive rdf:ID="ZeropositivemaximumPositive2">
                                <qr:hasInequality>
                                    <qr:SmallerThan>
                                <qr:hasInequalityTarget rdf:resource="&qrm;ZeropositivemaximumMaximum2" />
                                    </qr:SmallerThan>
                                </qr:hasInequality>
                                <qr:hasInequality>
                                    <qr:GreaterThan>
                                        <qr:hasInequalityTarget rdf:resource="&qr;Zero" />
                                    </qr:GreaterThan>
                                </qr:hasInequality>
                            </qr:Positive>
                        </qr:containsQualitativeValue>
                        <qr:containsQualitativeValue>
                            <qr:Maximum rdf:ID="ZeropositivemaximumMaximum2" />
                        </qr:containsQualitativeValue>
                    </qrm:Zeropositivemaximum>
                </qr:hasQuantitySpace>
            </qr:Magnitude>
        </qr:hasMagnitude>
        <qr:hasDerivative>
            <qr:Derivative rdf:ID="DerivativePressure1">
                <qr:hasQuantitySpace>
                    <qrm:Mzp rdf:ID="Mzp2">
                        <qr:containsQualitativeValue rdf:resource="&qr;Zero" />
                        <qr:containsQualitativeValue>
                            <qr:Min rdf:ID="MzpMin2">
                                <qr:hasInequality>
                                    <qr:SmallerThan>
                                        <qr:hasInequalityTarget rdf:resource="&qr;Zero" />
                                    </qr:SmallerThan>
                                </qr:hasInequality>
                            </qr:Min>
                        </qr:containsQualitativeValue>
                        <qr:containsQualitativeValue>
                            <qr:Plus rdf:ID="MzpPlus2">
                                <qr:hasInequality>
                                    <qr:GreaterThan>
                                        <qr:hasInequalityTarget rdf:resource="&qr;Zero" />
                                    </qr:GreaterThan>
                                </qr:hasInequality>
                            </qr:Plus>
                        </qr:containsQualitativeValue>
                    </qrm:Mzp>
                </qr:hasQuantitySpace>
            </qr:Derivative>
        </qr:hasDerivative>
    </qrm:Pressure>
```

```
</qr:hasConsequence>
<qr:hasConsequence>
    <qrm:Volume rdf:ID="Volume1">
        <qr:hasCausalDependency>
            <qr:PositiveProportionality>
                <qr:hasCausalDependencyTarget rdf:resource="&qrm;Height1" />
            </qr:PositiveProportionality>
        </qr:hasCausalDependency>
        <qr:hasMagnitude>
            <qr:Magnitude rdf:ID="MagnitudeVolume1">
                <qr:hasQuantitySpace>
                    <qrm:Zeropositivemaximum rdf:ID="Zeropositivemaximum3">
                        <qr:containsQualitativeValue>
                            <qr:Zero rdf:ID="ZeropositivemaximumZero3" />
                        </qr:containsQualitativeValue>
                        <qr:containsQualitativeValue>
                            <qr:Positive rdf:ID="ZeropositivemaximumPositive3">
                                <qr:hasInequality>
                                    <qr:SmallerThan>
                                <qr:hasInequalityTarget rdf:resource="&qrm;ZeropositivemaximumMaximum3" />
                                    </qr:SmallerThan>
                                </qr:hasInequality>
                                <qr:hasInequality>
                                    <qr:GreaterThan>
                                        <qr:hasInequalityTarget rdf:resource="&qr;Zero" />
                                    </qr:GreaterThan>
                                </qr:hasInequality>
                            </qr:Positive>
                        </qr:containsQualitativeValue>
                        <qr:containsQualitativeValue>
                            <qr:Maximum rdf:ID="ZeropositivemaximumMaximum3" />
                        </qr:containsQualitativeValue>
                        <qr:hasCorrespondence>
                            <qr:QuantityCorrespondence rdf:ID="Correspondence2">
                                <qr:hasCorrespondenceTarget rdf:resource="#Zeropositivemaximum1" />
                            </qr:QuantityCorrespondence>
                        </qr:hasCorrespondence>
                    </qrm:Zeropositivemaximum>
                </qr:hasQuantitySpace>
            </qr:Magnitude>
        </qr:hasMagnitude>
        <qr:hasDerivative>
            <qr:Derivative rdf:ID="DerivativeVolume1">
                <qr:hasQuantitySpace>
                    <qrm:Mzp rdf:ID="Mzp3">
                        <qr:containsQualitativeValue rdf:resource="&qr;Zero" />
                        <qr:containsQualitativeValue>
                            <qr:Min rdf:ID="MzpMin3">
                                <qr:hasInequality>
                                    <qr:SmallerThan>
                                        <qr:hasInequalityTarget rdf:resource="&qr;Zero" />
                                    </qr:SmallerThan>
                                </qr:hasInequality>
                            </qr:Min>
                        </qr:containsQualitativeValue>
                        <qr:containsQualitativeValue>
                            <qr:Plus rdf:ID="MzpPlus3">
                                <qr:hasInequality>
                                    <qr:GreaterThan>
                                        <qr:hasInequalityTarget rdf:resource="&qr;Zero" />
                                    </qr:GreaterThan>
                                </qr:hasInequality>
                            </qr:Plus>
```

```
                        </qr:containsQualitativeValue>
                    </qrm:Mzp>
                </qr:hasQuantitySpace>
            </qr:Derivative>
        </qr:hasDerivative>
    </qrm:Volume>
</qr:hasConsequence>
</qr:ModelFragment>
```

C.8 Scenario Definitions

```
<qr:Scenario rdf:ID="Connectedwatercontainerslefthasmorewater">
    <qr:hasFact>
        <qrm:Tube rdf:ID="Tube5" />
    </qr:hasFact>
    <qr:hasFact>
        <qrm:Container rdf:ID="Container5">
            <qr:hasConfiguration>
                <qrm:Connected rdf:ID="Connected5">
                    <qr:hasConfigurationTarget rdf:resource="&qrm;Tube5" />
                </qrm:Connected>
            </qr:hasConfiguration>
            <qr:hasConfiguration>
                <qrm:Contains rdf:ID="Contains5">
                    <qr:hasConfigurationTarget rdf:resource="&qrm;Oil5" />
                </qrm:Contains>
            </qr:hasConfiguration>
        </qrm:Container>
    </qr:hasFact>
    <qr:hasFact>
        <qrm:Container rdf:ID="Container6">
            <qr:hasConfiguration>
                <qrm:Connected rdf:ID="Connected6">
                    <qr:hasConfigurationTarget rdf:resource="&qrm;Tube5" />
                </qrm:Connected>
            </qr:hasConfiguration>
            <qr:hasConfiguration>
                <qrm:Contains rdf:ID="Contains6">
                    <qr:hasConfigurationTarget rdf:resource="&qrm;Oil6" />
                </qrm:Contains>
            </qr:hasConfiguration>
        </qrm:Container>
    </qr:hasFact>
    <qr:hasFact>
        <qrm:Oil rdf:ID="Oil5">
            <qr:hasQuantity>
                <qrm:Height rdf:ID="Height5">
                    <qr:hasInequality>
                        <qr:GreaterThan>
                            <qr:hasInequalityTarget rdf:resource="#Height6" />
                        </qr:GreaterThan>
                    </qr:hasInequality>
                    <qr:hasMagnitude>
                        <qr:Magnitude rdf:ID="MagnitudeHeight5">
                            <qr:hasQuantitySpace>
                                <qrm:Zeropositivemaximum rdf:ID="Zeropositivemaximum5">
                                    <qr:containsQualitativeValue rdf:resource="&qr;Zero" />
                                    <qr:containsQualitativeValue>
                                        <qr:Positive rdf:ID="ZeropositivemaximumPositive5">
                                            <qr:hasInequality>
                                                <qr:SmallerThan>
                                                    <qr:hasInequalityTarget
                                                        rdf:resource="&qrm;ZeropositivemaximumMaximum5" />
```

```
                                        </qr:SmallerThan>
                                    </qr:hasInequality>
                                <qr:hasInequality>
                                    <qr:GreaterThan>
                                        <qr:hasInequalityTarget rdf:resource="&qr;Zero" />
                                    </qr:GreaterThan>
                                </qr:hasInequality>
                            </qr:Positive>
                        </qr:containsQualitativeValue>
                        <qr:containsQualitativeValue>
                            <qr:Maximum rdf:ID="ZeropositivemaximumMaximum5" />
                        </qr:containsQualitativeValue>
                    </qrm:Zeropositivemaximum>
                </qr:hasQuantitySpace>
            </qr:Magnitude>
        </qr:hasMagnitude>
        <qr:hasDerivative>
            <qr:Derivative rdf:ID="DerivativeHeight5">
                <qr:hasQuantitySpace>
                    <qrm:Mzp rdf:ID="Mzp5">
                        <qr:containsQualitativeValue rdf:resource="&qr;Zero" />
                        <qr:containsQualitativeValue>
                            <qr:Min rdf:ID="MzpMin5">
                                <qr:hasInequality>
                                    <qr:SmallerThan>
                                        <qr:hasInequalityTarget rdf:resource="&qr;Zero" />
                                    </qr:SmallerThan>
                                </qr:hasInequality>
                            </qr:Min>
                        </qr:containsQualitativeValue>
                        <qr:containsQualitativeValue>
                            <qr:Plus rdf:ID="MzpPlus5">
                                <qr:hasInequality>
                                    <qr:GreaterThan>
                                        <qr:hasInequalityTarget rdf:resource="&qr;Zero" />
                                    </qr:GreaterThan>
                                </qr:hasInequality>
                            </qr:Plus>
                        </qr:containsQualitativeValue>
                    </qrm:Mzp>
                </qr:hasQuantitySpace>
            </qr:Derivative>
        </qr:hasDerivative>
    </qrm:Height>
</qr:hasQuantity>
</qrm:Oil>
</qr:hasFact>
<qr:hasFact>
<qrm:Oil rdf:ID="Oil6">
    <qr:hasQuantity>
        <qrm:Height rdf:ID="Height6">
            <qr:hasMagnitude>
                <qr:Magnitude rdf:ID="MagnitudeHeight6">
                    <qr:hasQuantitySpace>
                        <qrm:Zeropositivemaximum rdf:ID="Zeropositivemaximum6">
                            <qr:containsQualitativeValue rdf:resource="&qr;Zero" />
                            <qr:containsQualitativeValue>
                                <qr:Positive rdf:ID="ZeropositivemaximumPositive6">
                                    <qr:hasInequality>
                                        <qr:SmallerThan>
                                            <qr:hasInequalityTarget
                                                rdf:resource="&qrm;ZeropositivemaximumMaximum6" />
                                        </qr:SmallerThan>
```

```
                                        </qr:hasInequality>
                                    <qr:hasInequality>
                                        <qr:GreaterThan>
                                            <qr:hasInequalityTarget rdf:resource="&qr;Zero" />
                                        </qr:GreaterThan>
                                    </qr:hasInequality>
                                </qr:Positive>
                            </qr:containsQualitativeValue>
                            <qr:containsQualitativeValue>
                                <qr:Maximum rdf:ID="ZeropositivemaximumMaximum6" />
                            </qr:containsQualitativeValue>
                        </qrm:Zeropositivemaximum>
                    </qr:hasQuantitySpace>
                </qr:Magnitude>
            </qr:hasMagnitude>
            <qr:hasDerivative>
                <qr:Derivative rdf:ID="DerivativeHeight6">
                    <qr:hasQuantitySpace>
                        <qrm:Mzp rdf:ID="Mzp6">
                            <qr:containsQualitativeValue rdf:resource="&qr;Zero" />
                            <qr:containsQualitativeValue>
                                <qr:Min rdf:ID="MzpMin6">
                                    <qr:hasInequality>
                                        <qr:SmallerThan>
                                            <qr:hasInequalityTarget rdf:resource="&qr;Zero" />
                                        </qr:SmallerThan>
                                    </qr:hasInequality>
                                </qr:Min>
                            </qr:containsQualitativeValue>
                            <qr:containsQualitativeValue>
                                <qr:Plus rdf:ID="MzpPlus6">
                                    <qr:hasInequality>
                                        <qr:GreaterThan>
                                            <qr:hasInequalityTarget rdf:resource="&qr;Zero" />
                                        </qr:GreaterThan>
                                    </qr:hasInequality>
                                </qr:Plus>
                            </qr:containsQualitativeValue>
                        </qrm:Mzp>
                    </qr:hasQuantitySpace>
                </qr:Derivative>
            </qr:hasDerivative>
        </qrm:Height>
    </qr:hasQuantity>
    </qrm:Oil>
</qr:hasFact>
<qr:hasFact>
    <qr:hasValue rdf:ID="hasValue1">
        <qr:hasInequalityTarget rdf:resource="&qrm;ZeropositivemaximumPositive5" />
    </qr:hasValue>
</qr:hasFact>
<qr:hasFact>
    <qr:hasValue rdf:ID="hasValue5">
        <qr:hasInequalityTarget rdf:resource="&qrm;ZeropositivemaximumPositive6" />
    </qr:hasValue>
</qr:hasFact>
</qr:Scenario>
```