OWL and Qualitative Reasoning Models

Jochem Liem and Bert Bredeweg

Human Computer Studies Laboratory, Informatics Institute,
Faculty of Science, Universiteit van Amsterdam, The Netherlands
{jliem,bredeweg}@science.uva.nl

Abstract. The desire to share and reuse knowledge has led to the establishment of the Web Ontology Language (OWL) knowledge representation language. The Naturnet-Redime project needs to share qualitative knowledge models of issues relevant to sustainable development and OWL seems the obvious choice for representing such models to allow search and other activities relevant to sharing knowledge models. However, although the design choices made in OWL are properly documented, their implications for Artificial Intelligence (AI) are part of ongoing research. This paper 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. A parser has been developed to export (and import) the QR representations to (and from) OWL representations. To create the OWL definitions of the QR vocabulary and models, existing OWL patterns were used as much as possible. However, some new patterns, and pattern modifications, had to be developed in order to represent the QR vocabulary and models using OWL.

1 Introduction

During the development of the Web Ontology Language (OWL), design choices have been made 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. Another feature of OWL is that is has an open world assumption. The implications of these design choices on ontology development are still unclear, particularly for advanced applications in Artificial Intelligence (AI). The question is: *"What are the consequences of the OWL design choices on the expressiveness of the language for advanced AI applications?"*

To discover the problems and solutions associated with use of OWL, Garp3 Qualitative Reasoning (QR) models and their vocabulary [3] are formalised. Garp3 unifies three alternative approaches to qualitative reasoning (QPT [5], Envision [4], and QSIM [9]) into a single qualitative reasoning and modelling workbench. 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

C. Freksa, M. Kohlhase, and K. Schill (Eds.): KI 2006, LNAI 4314, pp. 33–48, 2007.

representation 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 which is understandable for non-experts, closely following the naive physics proposal (except for the focus on implementation) [7]. This common-sense view on systems which QR models have, make them interesting for reuse. Fourthly, the OWL community proposes that using an ontology as an information model for design is a typical use case [8]. Since both modelling and design are similar synthesis tasks [13], the formalisation of the QR domain should be a typical application of OWL. This makes the problems discovered during the formalisation of the QR application area of interest to a large group of researchers. Finally, there is a desire within the QR community to share and reuse models through a central online repository (see Figure 1). This goal will be realised within the European NaturNet-Redime project (http://www.naturnet.org/). A requirement to allow users to search for models in which specific concepts or structures are used, is a formalisation of these models in an open semantic format which is processable by query languages. For this purpose, OWL has been chosen since it is the de-facto standard for exchanging ontological models on the web, and has a large user base. Furthermore, well-developed OWL tools are available to facilitate the modelling and model search. Hence, this paper focuses on the question: *"Is OWL expressive enough to formalise the QR vocabulary and models, and which of the QR problems can an OWL reasoner solve?"*

In Figure 1 an overview is given of the desired result of this research. Traditionally, Garp3 can write models to a binary format and can also read them. Functionality has been added to export models to an OWL format and import them again. The models in the OWL format reference to model ingredients defined in the Qualitative Reasoning Vocabulary Ontology. For brevity, in this extra OWL file only the model itself is represented, as the simulations can be

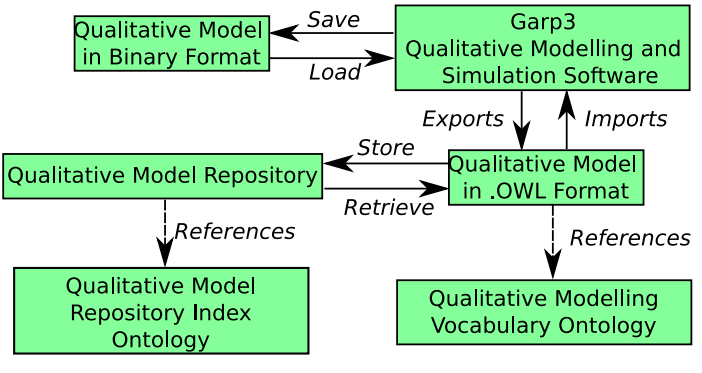


Fig. 1. Garp3's interaction with the binary and OWL files and the qualitative model repository

easily recreated using the software. In the near future an online model repository will be developed in which models can be stored, searched for and retrieved. Models in this repository will reference concepts in an ontology describing the categories within the repository.

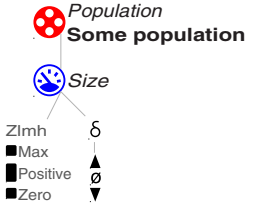
The organisation of this paper is as follows. Section 2 explains the QR field and the types of reasoning used. Section 3 describes the implications of the formalisation of general situations using QR ingredients on the use of the OWL reasoner. Section 4 focuses on the reusability of reified relations. Section 5 explains multiple methods of the formalisation of a total order of values. Section 6 describes a pattern to restrict the use of relations for classes with specific conditions. Finally, the results are discussed and conclusions are drawn.

2 Qualitative Reasoning

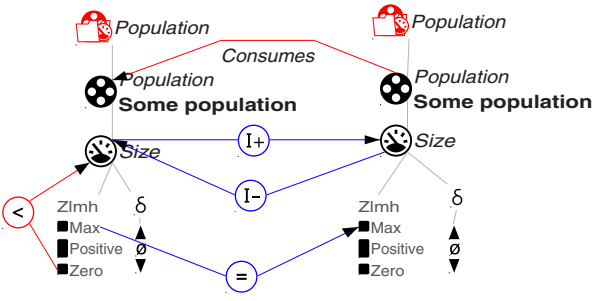
The aim of qualitative modelling and reasoning [3] is to build models from which the behaviour of systems (in the form of state graphs such as the one in Figure 5a) can be predicted through simulation. Each state describes a specific situation of the system, while each transition represents the changes from one situation to another. QR models require no numerical data. Instead, changeable properties of systems are described as its relevant points and intervals (see Figure 2a and section 5). The size of a population in an environment can be formalised as {zero, positive, max}. This kind of formalisation is particularly advantageous for domains in which it is difficult to obtain numerical data, such as ecology. For experts in these fields, qualitative modelling provides a means to make their knowledge explicit and computer processable. An example of the application of qualitative modelling in ecology is the testing of the succession hypothesis of the Brazilian Cerrado forest [12]. A detailed description of the application of QR in ecology is available in the Ecological Informatics book [2].

An advantage of qualitative modelling is that the causal dependencies between quantities are made explicit (the [I]nfluence and [P]roportionalities in Figure 2b). Next to the ability to predict the behaviour of a system, these causal dependencies make it possible to provide a causal explanation of why a system behaves in a particular way. These features of qualitative simulation provide the opportunity for hypothesis testing and learning.

An important part of a QR model are model fragments, which incorporate model ingredients as either conditions (red) or consequences (blue). Two example model fragments can be seen in Figure 2a and 2b. The first describes a population with a size (which can be read as: if there is a population, it has a size), while the second formalises the consumption process between two populations. In general, the structure of the system is described using conditions and the causal dependencies as consequences, although other model ingredients can also be used as consequences. The model fragment describes causal relations which apply to a general situation within a system. Model fragments are organized in a subtype hierarchy. A child model fragment inherits the model ingredients from their parent and is a specialisation of its parent, because it adds



(a) A single population with size zero (point), positive (interval) or maximum (point). The population entity is conditional, while the size quantity is a consequence.



(b) The consumption of a prey by a predator. The population model fragments, the consumes configuration and the size greater than zero inequality are conditional. The causal dependencies and the equality between the max values are consequences.

Fig. 2. The two model fragments in the example model

new model ingredients to the aggregate. It is possible to reuse a model fragment within another model fragment (in Figure 2b the *Population* model fragment is reused twice). Technically speaking, this is similar to the relation between a parent and a child model fragment: the model ingredients of the reused model fragment are incorporated as conditions in the model fragment. This type of reuse allows users to efficiently create qualitative models.

Scenarios are the counterpart of model fragments and describe specific situations of systems (see Figure 3). These aggregates are used to determine the start states of the behavioural graph.

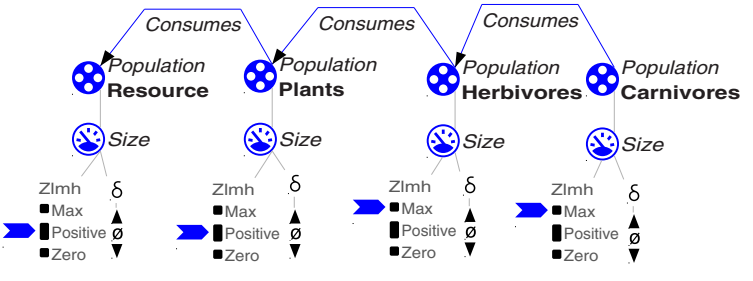


Fig. 3. A scenario plants consuming a resource, herbivores consuming plants, and carnivores consuming herbivores. All the model ingredients are consequences (facts).

The reasoning the QR engine performs can be divided into five parts: classification, inequality testing, consequence merging, influence resolution and prediction.

The first four steps take an incomplete state (which, in the first algorithm iteration, is the scenario) as input and produce a complete state description, i.e. a state containing all consequences of the model fragments applying and with calculated derivatives. The classification task searches for candidate model fragments. Candidate model fragments are the model fragments which structurally match the incomplete state. The behavioural aspects of model fragments (such as known values and inequalities) are ignored when searching for model fragments, as model fragments can contain inequalities as conditions. These conditional inequalities might be true, but have to be derived from the other inequalities which also apply to the state (which is not part of the classification).

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. The reasoner tries to derive the conditional inequalities in the inequality testing step. If the inequalities can be deduced, they are incorporated in the state. If the inequalities are inconsistent with the established inequalities, the candidate model fragment is removed. Model fragments for which both the conditional structure (which can include other model fragments) and the conditional values and inequalities match, become *active*. In the consequence merging step, the consequence model ingredients of active model fragments are added to the state.

The classification, inequality testing and consequence merging steps are repeated with the augmented state until no new applying model fragments can be found. After these steps, candidates are either (1) included in the state because their conditions are true, (2) ignored because their conditions are inconsistent with the state. This results in an augmented state, which incorporates all the consequences of matching model fragments. Simulating the scenario in Figure 3 with the model fragments from Figures 2a and 2b would result in an augmented state as visualised in Figure 4, although the derivatives would still be unknown (the arrows next to the active values indicating the trend).

In the influence resolution step the augmented state is completed by determining the derivatives of the quantities by resolving the influences and proportionalities. Influences are the cause of change within a model, and are therefore said to model processes. Depending on the magnitude value of the source quantity and the type of influence, the derivative of the target quantity either increases or decreases. An influence Q1(I+)Q2 causes the quantity Q2 to increase if Q1 is positive, decrease if it is negative, and remain stable when it is zero (assuming there are no other causal dependencies on Q2). For an influence I- this is just the opposite. Influences are also referred to as direct influences. Proportionalities 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 Q1(P+)Q2 causes Q2 to increase if Q1 increases, decrease if Q1 decreases, and remain stable if Q1 remains stable. For a proportionality P- the opposite applies. Applying the influence resolution step would result in the completed state description shown in Figure 4.

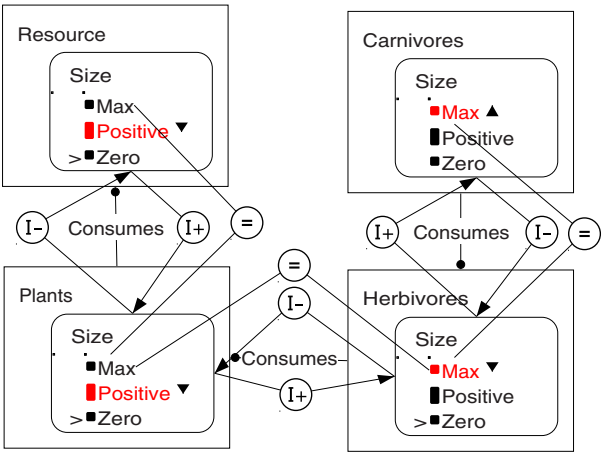


Fig. 4. The completed state description after matching model fragments on the scenario, aggregating the consequences and resolving the causal operators

The prediction algorithm takes the completed state description and identifies the successive states of behaviour and transitions to them. Termination rules are part of the qualitative engine, and indicate under what conditions states change. For example, if the magnitude of a quantity is at a point-value (e.g. the population size is zero), and the derivative of that quantity is positive (e.g. the population size is increasing), then in the next state that quantity has the interval-value directly above its current point-value (e.g. the population size becomes positive). Using this set of rules all the possible terminations of the state are gathered. Not every termination in the set of possible terminations of a state applies. Some terminations 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. Others occur simultaneously due to correspondences. The final step generates the successive states and transitions using the final set of pruned and merged terminations. For these successive states all the algorithm steps are repeated to generate a complete state graph describing the behaviour of the modelled system.

The state graph resulting from the simulation of the scenario in Figure 3 is shown in Figure 5a. The values of the quantities in each of the states are shown in Figure 5b. The transitions from state 1 to 2 and 3 happen because there is a negative influence (consumption) on the size of the herbivore population which is greater than the positive influence on the herbivores (feeding), since the magnitude of plant population size is smaller than the magnitude of the carnivore population size. This transition has priority over all other transitions because it is a change from a point to an interval, while the other possible transitions are changes from intervals to points. The reason that two states are generated is because the influences on plant population can become either equal, resulting in a stable derivative for the plant population (state 3), or unequal, resulting in

the decrease of the plant population size. From state 2, three possible changes may occur. Either only the resource depletes (state 4), both the resource depletes and the plant population becomes zero (state 5), or the resource depletes and the plant and the herbivore populations die out (state 6). From state 3 it is only possible that the resource depletes (state 4), as the derivative of the plant population is stable. As a result of the depletion of the resource, the plant population decreases again, because the positive influence on the plant population from feeding disappears. The other transitions are obvious. From state 4, either both the plant and the herbivore populations die instantly (state 6), or the plant population dies first (state 5), and the herbivore population becomes extinct afterwards (state 6).

(a) The state graph resulting from simulation. **(b)** The value history corresponding to states in the state graph.

Fig. 5. The state graph and corresponding value history

3 Representing General Situations

The reasonable aim of formalising QR in OWL would be to try to use an OWL classifier to solve the classification task instead of the QR reasoner. Taking that approach, some of the typical inferences made by the QR reasoner such as the inequality testing, consequence merging, influence resolution and prediction tasks, would then still be left to the QR reasoner. To fulfil this 'reasonable' goal the correct formalisation of model fragments is essential. However, as will be pointed out below, this is already rather complex and not adequately solvable with the current version of OWL [1].

The conditional model ingredients in model fragments describe general situations of a system, and scenarios describe specific situations. The classes in OWL describe general concepts, while the instances are specific individuals. OWL reasoners are able to classify instances to classes, therefore the model fragments have to be formalised as classes and the scenarios as instances. As a result the contents of the model fragments has to be formalised using necessary and sufficient conditions. This allows the scenarios to be classified as a certain model fragment. The consequences of model fragments have to be modelled separately, as they cannot be part of the restrictions of the model fragments (they are the consequences of a model fragment firing).

The formalisation of model fragments as classes allows the representation of the subtype hierarchy of model fragments. The subclasses of the model fragments would inherit the restrictions which model the contents of the parents, and add restrictions to describe the new model ingredients of the child model fragment.

There are two problems with the formalisation of the conditions of model fragments as classes. Firstly, it is impractical to specify that a model fragment contains multiple objects of the same type (either as conditions or consequences). It would require two separate relations (one for conditions and one for consequences) for each type of object in combination with a cardinality restriction (to indicate the number of incorporated objects of that type). The unique relations for each object type are required as the cardinality restriction has to apply to precisely one type of object. This problem can be solved by making use of Qualified Cardinality Restrictions (QCR), which indicate a class should have a certain amount of fillers for a specific relation. Although QCR's are not in the first OWL specification, Protégé [6] and RacerPro (Racer Systems GmbH & Co. KG: http://www.racer-systems.com) have already added support for them[1]. Furthermore, QCR's are already mentioned in the drafts of the OWL1.1 specifications[2].

The second problem concerns the impossibility to distinguish between different objects of the same type in restrictions. Consider population x consuming population y, which in turn consumes z (a situation similar to the one described in Figure 3 if it was a model fragment with conditional elements). When formalised as restriction in OWL, this would result in the following definition:

hasCondition exactly 3 *Population*
hasCondition some (*Population and*
(*consumes some* (*Population and*
(*consumes some Population*))))

This formalization has multiple interpretations. It could be that population x preys on y, and y preys on x, or that x preys on y, and y preys on z. Furthermore, it becomes hard to formalize the other relations the populations take part in without *naming* the Populations. Without variables to distinguish between two objects of the same type, it is impossible to describe model fragments as classes.

As OWL is not expressive enough to formalize model fragments as classes, there is no choice but to formalize them as instances. This makes it impossible to use an OWL reasoner to classify scenarios on model fragments. On the other hand, using instances does eliminate the requirement of having to separate the conditions and consequences in the formalization of model fragments. Each model fragment can be modelled as an instance. That instance has hasCondition and hasConsequence relations to each of the model ingredients instances it contains. Those ingredient instances in turn have relations which indicate how they are related.

[1] http://protege.stanford.edu/mail_archive/msg17798.html
[2] http://owl1_1.cs.manchester.ac.uk/

A problem with the formalization of model fragments as instances is that model fragments can have subclasses and can be reused. Since it is impossible to create instances of instances or subclasses of instances, model fragments cannot be described as instances.

Summarising, model fragments have to be modelled as classes in order to represent the subtype hierarchy of model fragments, and to reuse them in other model fragments. On the other hand, classes are not expressive enough to model the contents of model fragments. Secondly, in order to correctly formalize the contents of model fragments they have to be modelled as instances. However, this would make it impossible to keep track of the reuse of model fragments, and would require a special subclass relation to formalize the model fragment hierarchy.

Both previous results are undesirable. An alternative is to treat model fragment classes as individuals. This is valid, but makes the ontology OWL Full [1]. 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 restrictions in OWL. However, the solution does correctly represent model fragments and will allow the OWL format to be used for model search and reuse.

4 The Formalisation of Relations

The relations in the QR vocabulary, the *configurations* and *dependencies* (causal, mathematical and correspondence), are not simple binary relations between two objects. For each relation screen information (for visualisation), such as its position, has to be stored. Additionally, the *Calc* relations plus and min potentially link more than two objects, as they model the result of an addition or subtraction of two values, and can have an arbitrary amount of *inequality* relations to compare the result to other values (e.g. it is possible to specify that the sum of the two population sizes in Figure 2b is equal to zero). Since OWL supports only binary relations between objects, the formalisation of n-ary relations and information about relations is an issue. The Semantic Web Best Practices and Deployment Working Group (http://www.w3.org/2001/sw/BestPractices/) describes two variations of a pattern which can be applied to solve this issue [10]. By modelling a relation as a class it is possible to relate multiple objects using only one relation instance. This process is called reification.

However, the existing reification pattern hinders the reuse of reified relations. The calc and inequality relations in the QR vocabulary may only connect specific subsets of elements of quantities (magnitudes, derivatives and points) and other calc relations. For example, magnitudes can only be connected to other magnitudes or points fulfilling certain conditions, while derivatives can only be connected to derivatives or points fulfilling certain other conditions (see Section 6). Thus, the second argument of the relation depends on the first one, but

the relations have the same meaning independent of the arguments. It is important that the relations can be reused. Conceptually, there are 3 Calc relations (Plus, Min and their superclass), and 6 inequalities ($<, \leq, =, \geq, >$ and their superclass) in the QR vocabulary. The need to create multiple types of Calc and Inequality relations depending on their arguments creates unwanted complexity in the file format and the QR vocabulary ontology.

If the first variation of the pattern is applied to represent the use of Calc relations by magnitudes, magnitudes are restricted to having an arbitrary amount of Calc relations (*hasCalc only Calc*). Furthermore, the Calc relations have to be related to exactly one magnitude and have an arbitrary amount of inequalities (*hasCalcTarget only Magnitude*; *hasCalcTarget* $= 1$; *hasInequality only Inequality*). This fixes the target of the Calc relation, making it impossible to reuse the relation for Calc relations between derivatives. At least the Calc relation can be reused by classes with the same target. If the restrictions for the inequalities from Calc relations are ignored, 6 different relations are needed to formalise all the restrictions. For the inequalities 24 reified relations are needed.

The second variation of the pattern only worsens the problem, as the restrictions are all formalised in the Calc relation. These indicate that both its source and its target have to be of a specific type (*hasCalcSource only Magnitude*; *hasCalcSource* $= 1$; *hasCalcTarget only Magnitude*; *hasCalcTarget* $= 1$). This prevents reuse of the relation for all other classes. To properly formalise the relation restrictions, 15 reified Calc relations and 36 reified inequality relations are needed.

We developed a new version of the reification pattern to solve the reusability issues of reified relations. Compared to the first version of the reification pattern, the restrictions about the target of the Calc relation and its Inequalities have been moved to the source of the relation (see Figure 6 which uses the Protégé [6] syntax). The formalisation represents that Magnitudes can have an arbitrary amount of hasCalc relations with reified Calc relations, which in turn have exactly one hasCalcTarget relation with another magnitude, and an arbitrary amount of inequality relations with magnitudes. This is achievable as OWL allows the creation of new anonymous class definitions within restrictions (the conjunction in the hasCalc restriction). Using this pattern, it is possible to impose specific usage restrictions for relations in each desired source class. For example, the derivative class can have an arbitrary amount of hasCalc relations with reified Calc relations, which have exactly one other derivative as a target, and an arbitrary amount of inequalities with other derivatives. Note that the real formalisation is more complex, as inequality properties are also reified, and the possible values of the hasCalcTarget and inequality properties are a union of multiple classes.

Another advantage of this new pattern is that it is possible for other users to use the relation defined in the ontology (without having to copy and adapt it), as the usage of the relation is not restricted to specific classes. This is also a disadvantage, as it is possible to abuse the relation with classes for which it does not make sense. A possible work-around is adding a cardinality=0 restrictions

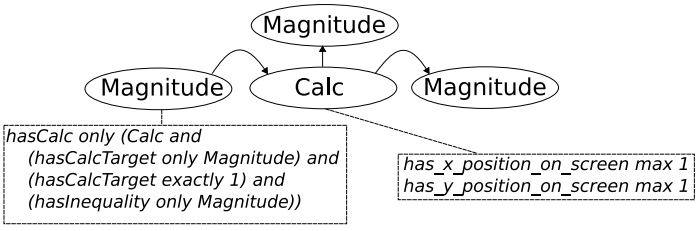


Fig. 6. A reusable reification pattern

to classes for each relation the class cannot be involved with, although this is only an acceptable solution if the number of relations and classes is relatively low.

5 Representing Values

Qualitative values are either points or intervals which are stored in quantity spaces (see Figure 2a). These behavioural ingredients define the possible values of a quantity. A quantity space consists of at least one qualitative value and values adjacent to intervals have to be points and visa versa. The quantity space describes a total order, which means that a magnitude or derivative can only change to a value directly above or below its current value. Qualitative values in a model fragment can participate in (in)equality, correspondence and calc relations. As a result, it is impossible to formalise them as an enumeration of individuals (the 'values as sets of individuals' pattern [11]), as different relations in different contexts would refer to the same value. Therefore, the formalization of qualitative values requires a unique individual for each value instance (i.e. for each quantity space in which the value occurs).

Representing the qualitative values as classes (the 'values as subclasses partitioning a "feature"' pattern [11]) fulfils our requirement of creating a unique value individual for each quantity space instance. The qualitative values can be thought of as a set of subclasses forming a parent class. This class is exactly the superset of all the possible value classes (modelled using *owl:unionOf*). It is necessary to explicitly state that the subclasses are disjoint, as it should be inconsistent to create an individual which is an instance of multiple values.

Representing qualitative values as classes allows individuals to be created for each instance of a specific quantity space. However, the pattern does not model their strict ordering. A possible solution to this problem is to model the values using an RDF collection [10]. Such a collection consists of a number of instances of *rdf:List*. Each of these items is connected to the next in the collection using *rdf:rest*, and points to a qualitative value instance using *rdf:first*. An advantage of this pattern is that OWL editors understand the that the structure is a list, as it is part of the RDF specification. A disadvantage is that using *rdf:List* causes the ontology to become OWL Full. This disadvantage can be remedied by recreating

a list structure in OWL [10], but a side effect would be that the editors would not understand that the structure modelled is a list.

The list pattern has two further problems. Firstly, a list has no ontological meaning, as it is a data structure. A list with the cities Amsterdam, Brussels, and Paris has little meaning. They could indicate a travel route, cities with around the same amount of inhabitants, or something else entirely. The relations between the list items is left implicit. Secondly, it becomes impossible to classify a list entry depending on the owner instance of the list, as there is no direct relation between that object and each list item. This would make the "relation restriction through classification" pattern impossible to apply (section 6).

Our solution to formalise quantity spaces makes the ordering of its values explicit using inequalities (see Figure 7). The quantity space is connected to its point and interval instances using *containsQualitativeValue* relations. The ordering is established using reified inequality relation instances originating from the intervals, as inequalities created by the user are required to originate from the points. Each consecutive value in the order must have another type than the previous one. Therefore, the intervals can only have inequalities with points. This solution is a semantic description of values, but OWL editors do not understand it is a list.

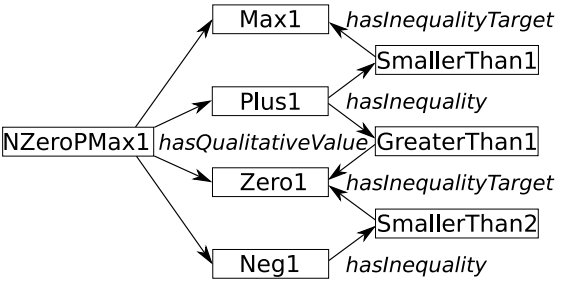


Fig. 7. Representing a quantity space and its values using inequalities

6 Relation Restriction Through Classification

A quantity consists of a magnitude and a derivative, which in turn each have a quantity space (the magnitude is not visualised in Figure 2a). As described above, quantity spaces consist of a set of values in a total order. Roughly speaking, inequalities are ordinal relations which have either magnitude or derivative items (values, calc relations and magnitudes and derivatives themselves) as arguments. Since manually categorising values into either a class for points belonging to magnitudes or a class for points belonging to derivatives would add redundant information to the formalisation, a different solution is desired.

Our inequality formalisation solution introduces two new classes, one for the point-values belonging to magnitudes (PointBelongingToMagnitude), and another for point-values belonging to derivatives (PointBelongingToDerivative).

In these classes necessary ($class \implies conditions$), and necessary and sufficient ($class \iff conditions$), restrictions are combined. The necessary and sufficient conditions of PointBelongingToMagnitude state that the value belongs to a quantity space which belongs to a magnitude (or a derivative for PointBelongingToDerivative). This allows the OWL reasoner to classify the points as belonging to one of the classes. The necessary conditions specify that all the inequality relations instances which have a PointBelongingToMagnitude as a first argument must have either a PointBelongingToMagnitude or a Calc relation between magnitude items as a second argument.

This pattern is used to restrict the use of inequalities, Calc relations and correspondences. Essentially, a new class is created with necessary and sufficient conditions for one of the special cases, and necessary conditions for the restriction of this special case. This pattern takes away the need to replicate information by using the OWL reasoner.

7 Implementation

We have developed a generic ontology [14] of the QR vocabulary in which all the model ingredients and their usage restrictions are formalised (see Figure 8). Based on this formalisation we have successfully implemented OWL export and import functionality, which has been integrated with the Garp3 qualitative reasoning and modelling tool (http://hcs.science.uva.nl/QRM/) using the SWI-Prolog Semantic Web Library [16]. This functionality is currently used to automatically formalise QR models as domain ontologies, which can be shared using an online model repository. The consistency of these model ontologies was checked using the Triple20 [15] and Protégé [6] ontology editors and the RacerPro reasoner. The QR vocabulary ontology and an example QR model in OWL can be found via: http://protege.cim3.net/cgi-bin/wiki.pl?NaturNet_Redime.

8 Conclusions and Discussion

This paper presented the problems encountered during the formalisation of the QR vocabulary and models in OWL. We succeeded in formalising qualitative models in OWL, allowing our formalisation of models to be used as a data format for a central model repository. Due to the limits of the expressiveness of OWL, it was not possible to formalise the model fragments as purely classes with restrictions. Instead, they were formalised as classes with relations to instances (making the ontology OWL Full). This makes it impossible to use the OWL reasoners to classify scenarios on model fragments, since the instances and relations are not necessary and sufficient conditions.

To be able to formalise n-ary relations and represent information about them the reification pattern is used. We have shown that the two existing variations prevent the relations to be reused. Our new pattern solves this reusability issue by moving the relation restrictions to the source class.

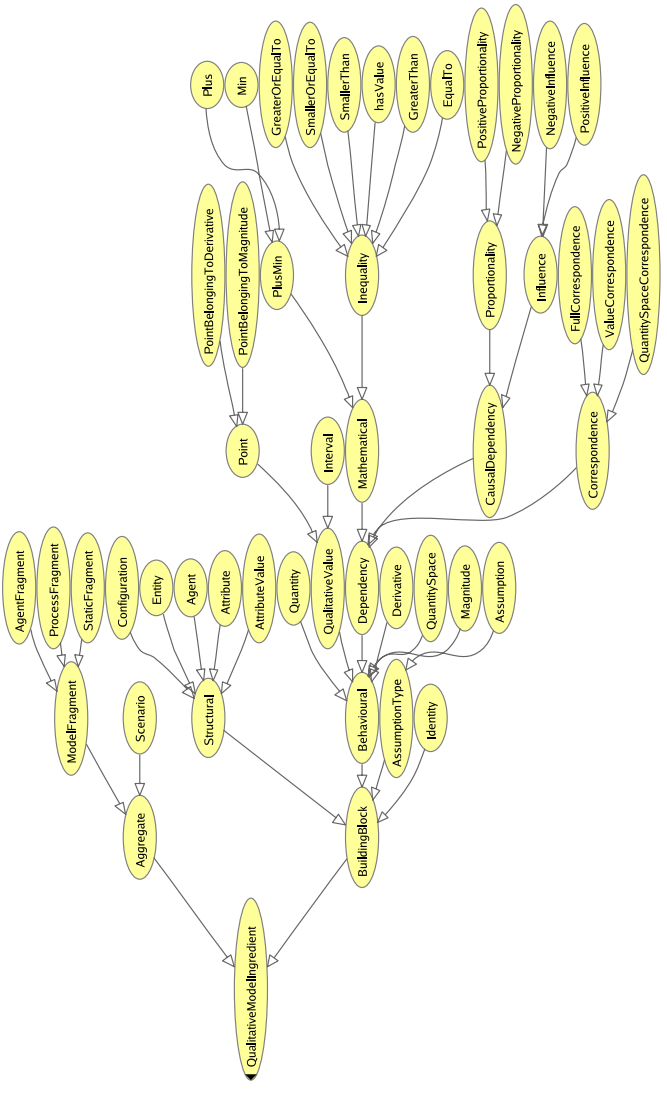


Fig. 8. The taxonomy of qualitative model ingredients

The existing patterns to formalise sequences of values are not enough to represent quantity spaces. The problems with representing them as a enumeration of individuals, a set of classes, and an *rdf:list* of instances were explained. We presented our more semantic representation which solves these problems.

We developed a new pattern to impose relation usage restrictions on classes with certain conditions. New classes are defined with the conditions as necessary and sufficient restrictions. The necessary restrictions are imposed on these classes, and will apply when instances are classified as belonging to the class. This prevents

information redundancy, as instances do not have to be explicitly represented as belonging to certain classes.

Future OWL extensions (such as the Semantic Web Rule Language) might make it possible to formalise the model fragments in such a way that it is possible to use an OWL reasoner to solve the QR classification task. For an OWL reasoner to actually replace a QR classification task such as found in Garp3, two additional inferences have to be addressed. Firstly, the reasoner tries to infer if certain conditional inequalities are true in the scenario using inequality reasoning. Secondly, unprovable but possible conditional inequalities and value assignments are assumed by the reasoner. Each mutually exclusive set of these assumptions will result in a new state in the state graph describing the behaviour of the system. Another inference which would be useful to OWL users is adding individuals to a knowledge base using a rule based mechanism. This would allow the replacement of the QR consequence merging task.

The research described in this paper has allowed the implementation of functionality to export qualitative models from the Garp3 application to an OWL file, and import this OWL file again into the workbench. This makes it possible to store these qualitative models in an online repository. This repository should make it possible for the community of practice to (1) share models amongst themselves, (2) search for models in which specific concepts or structures are used, and (3) reuse parts of models, which are all goals of the NaturNet-Redime project. Domain experts can use this repository to search for modelling work related to their own research, for example to compare different formalisations of the same phenomena. Teachers can ask students to download and analyse certain models. Finally, the repository allows modellers to store their results and disseminate them to a larger group.

Acknowledgements

This work is co-funded by the European Commission within the Sixth Framework Programme (2002-2006), project NaturNet-Redime (http://www.naturnet.org), number 004074. We would like to thank Rinke Hoekstra and the reviewers for their valuable comments, and Jan Wielemaker for his extensive programming support.

References

1. S. Bechhofer, F. van Harmelen, J. Hendler, I. Horrocks, D. L. McGuinness, P. F. Patel-Schneider, and L. A. Stein. OWL web ontology language reference. W3C recommendation, February 2004. M. Dean, G. Schreiber (eds.).
2. B. Bredeweg, P. Salles, and M. Neumann. *Ecological Informatics: Scope, Techniques and Applications*, chapter Ecological Applications of Qualitative Reasoning, pages 15–47. Springer, Berlin, 2nd edition, 2006.
3. B. Bredeweg and P. Struss. Current topics in qualitative reasoning (editorial introduction). *AI Magazine*, 24(4):13–16, 2003.

4. J. de Kleer and J. S. Brown. A qualitative physics based on confluences. *Artificial Intelligence*, 24(1-3):7–83, December 1984.
5. K. D. Forbus. Qualitative process theory. *Artificial Intelligence*, 24(1-3):85–168, December 1984.
6. N.F. Noy H. Knublauch, R. Fergerson and M.A. Musen. The protege owl plugin: An open development environment for semantic web applications. In S. A. McIlraith, D. Plexousakis, and F. van Harmelen, editors, *International Semantic Web Conference*, pages 229–243, Hiroshima, Japan, November 2004. Springer.
7. P. 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.
8. J. Heflin. OWL web ontology language use cases and requirements. W3C recommendation, February 2004.
9. B. Kuipers. Qualitative reasoning: modeling and simulation with incomplete knowledge. *Automatica*, 25(4):571–585, 1989.
10. N. Noy and A. Rector. Defining n-ary relations on the semantic web. W3C working group note, April 2006. http://www.w3.org/TR/swbp-n-aryRelations/.
11. A. Rector. Representing specified values in OWL: "value partitions" and "value sets". W3C working group note, May 2005. http://www.w3.org/TR/swbp-specified-values/.
12. P. Salles and B. Bredeweg. Qualitative reasoning about population and community ecology. *AI Magazine*, 24(4):77–90, 2003.
13. G. Schreiber, H. Akkermans, A. Anjewierden, R. de Hoog, N. Shadbolt, W. van de Velde, and B. Wielinga. *Knowledge Engineering and Management - The CommonKADS Methodology*. MIT Press, Cambridge, MA, 2000.
14. G. van Heijst, S. Falasconi, A. Abu-Hanna, G. Schreiber, and M. Stefanelli. A case study in ontology library contruction. *Artificial Intelligence in Medicine*, 7(3):227–255, June 1995.
15. J. Wielemaker, G. Schreiber, and B. Wielinga. Using triples for implementation: the Triple20 ontology-manipulation tool. In Y. Gil, E. Motta, V. R. Benjamins, and M. A. Musen, editors, *International Semantic Web Conference*, pages 773–785, Berlin, Germany, November 2005. Springer Verlag. LNCS 3729.
16. J. Wielemaker, G. Schreiber, and B. J. Wielinga. Prolog-based infrastructure for RDF: performance and scalability. In D. Fensel, K. Sycara, and J. Mylopoulos, editors, *The Semantic Web - Proceedings ISWC'03, Sanibel Island, Florida*, pages 644–658, Berlin, Germany, october 2003. Springer Verlag. LNCS 2870.