Project no.	**004074**
Project acronym:	**NATURNET-REDIME**
Project title:	**New Education and Decision Support Model for Active Behaviour in Sustainable Development Based on Innovative Web Services and Qualitative Reasoning**
Instrument:	**SPECIFIC TARGETED RESEARCH PROJECT**
Thematic Priority:	**SUSTDEV-2004-3.VIII.2.e**

D4.3 Collaborative QR model building and simulation workbench[1]

Due date of deliverable: **01/09/2006**
Actual submission date: **12/09/2006**

Start date of project: **1st March 2005** Duration: **30 months**

Organisation name of lead contractor for this deliverable:
UvA

Revision: Final

Project co-funded by the European Commission within the Sixth Framework Programme (2002-2006)		
Dissemination Level		
PU	Public	X
PP	Restricted to other programme participants (including the Commission	
RE	Restricted to a group specified by the consortium (including the Commission	
CO	Confidential, only for members of the consortium (including the Commission Services)	

[1] Authors: Jochem Liem, Anders Bouwer, Bert Bredeweg

Abstract

This document describes the functionality added to Garp3 to support the collaborative construction of qualitative models. It includes (1) a new environment called Sketch in which intermediate model results can be created; (2) OWL export- and import-functionality, so models can be exported to an OWL format, and models in that format can be imported again; (3) the model repository, which is a central place to store and search for qualitative models saved in the OWL format; (4) multiple model support to allow having multiple models open in the Garp3 workbench at the same time; and (5) model merging facilities, which make it possible to copy/paste parts of models into other models. All these features facilitate collaborative construction of conceptual models on issues relevant to sustainable development.

Document history

Version	Status	Date	Author
0.1	Copy Paste Usage	16-08-2006	Liem, J.
0.2	Sketch Environment, Rewritten Introduction	31-08-2006	Bouwer, A.
0.3	Restructured Document	31-08-2006	Liem, J.
0.4	Rewritten OWL support Added multiple model support Added model repository	01-09-2006	Liem, J.
0.5	Finalised Document (abstract, introduction, figures, conclusions, references)	01-09-2006	Liem, J.
0.6	Incorporated comment by Bert Bredeweg Improved Model Merging Extended introduction with requirements	04-09-2006	Liem, J.
0.7	Integrated final comments by Bert Bredeweg Replaced images with Windows screenshots	12-09-2006	Liem, J.

Contents

1. INTRODUCTION

This document describes the collaborative version of the qualitative modelling and reasoning workbench Garp3. It supports users in collaborating at different stages of the model building process as described in the design document for the collaborative workbench [9].

The approach taken for creating a QRM workbench that supports collaborative modeling is based on two perspectives. One concerns the idea that multiple users work together using a shared workbench. The other perspective focuses on individual users reusing model parts of other modelers. Preferably, the QRM workbench should support both approaches.

1.1. Workbench sharing

This is a synchronous mode of working, in which users share a single workbench and together create a model, e.g. using Garp3 while sitting behind the same desk and working with the same terminal. Alternatively, users could be physically remote from each other and share a single version of the workbench via tools such as VNC[2]. Although this mode of working is useful, other means to support collaboration are also crucial. In particular, users will often start with global, vague, possibly inconsistent and conflicting ideas, which will gradually develop into a mutually shared understanding while moving towards a detailed version of the model. Organizing and explicating this process in software is a better means of support collaboration between modelers, as it allows them to negotiate in order to understand and solve inconsistencies between models and model parts.

1.2. The role of intermediate model results

The intermediate model results presented in the structured approach to modeling [1,12] provide handles to enable and support both synchronous (shared modeling) and asynchronous (model reuse) collaborative working. Following this ‘Framework for conceptual QR description of case studies’ a distinction can be made between high-level and low-level aspects of a model. High-level corresponds to less specific, more general, whereas low-level refers to specific and detailed. A concept map can be considered to be at a higher level than a causal model. A causal model is at a higher level than a full model with simulation results. High-level aspects are usually easier to understand, exactly because they require less specific information. On the other hand, they leave matters open, underspecified. Obviously, a better and more detailed understanding goes together with transforming less detailed model results into more detailed ones.

The basic idea behind the approach for creating the QRM workbench for collaboration is that collaboration is facilitated by an interconnected web of multiple sets of higher-level concepts in addition to the ultimate domain and model specific terms:

- Higher-level details are less specific, and participants will find it easier to understand each other and adapt to alternative views.

[2] http://www.tightvnc.com

- Changes to model details can be done more easily, and therefore models can at that level be aligned more easily.
- When a detailed model is available, having high-level connections to those detailed model ingredients provides leverage to compare alternative detailed models. There are more labels available to analyze and compare models, and the details at a lower level may point to similar high-level concepts, providing guidance in comparing models and enabling model merger.
- High-level model results can be used as an umbrella under which a whole cluster of detailed and potentially rather distinct models reside. This could reflect a community in which stakeholders share a high-level description of some phenomenon, but differ on the details as reflected by the individual models derived from the shared understanding.

The above leads to the following conclusion. To support collaborative modeling it is important to have explicit access to intermediate model results at different levels of detail in addition to the ultimate model and its simulation results. The ‘Framework for conceptual QR description of case studies’ [1,12] provides an important starting point for defining such intermediate model results.

A new environment was developed for creating such intermediate model results as conceptual sketches: the Sketch Environment. It allows users to make their conceptual ideas explicit in intermediate ‘sketch’ representations, without being explicitly bound to the QR vocabulary and constraints. It consists of seven new editors for specifying a concept map, structural model, causal model, process definitions, actions and external influences, scenarios, and behaviour graphs. Collaborating users can share these intermediate representations to facilitate the development of a common understanding of the domain subject before actually implementing the model. In addition, it allows users to acquire a basic understanding of a model, without having to analyse the detailed model results. The Sketch environment is described in Chapter 2.

1.3. Reuse of models by single users

For the asynchronous mode of working users should be allowed to save models to external files and store them in a repository. Other users should be given facilities to search the repository in order to find models, or parts of models, that fit their interests. When reusing existing models, different options exist. One option is to refine, modify, or extend the reused model so that it better fits the user’s needs. Another option is to merge the reused model, or model parts, with model details already created by the user.

The storage of models in a repository is only useful if the repository can be searched in a user-friendly way. Automated search facilities for models based on model content used to be impossible, since models were stored in a binary format. Therefore, functionality has been added to Garp3 to export models to a Web Ontology Language (OWL) file, and import them again. Models formalised in OWL can be stored in a central repository. The OWL semantics (and the available tools which implement this semantics) allows searching for models in which specific concepts are used. The OWL functionality is described in Chapter 3.

With models formalised in OWL it is possible to develop an online qualitative model repository. This functionality is not integrated with Garp3 itself, but implemented as a web application interacting with the OWL files generated by the Garp3 OWL export functionality. Users are able to store their models in the model repository, and can search for models about specific domain subjects. This allows a community to efficiently collaborate, since they have access to earlier work. The model repository is described in Chapter 4.

To allow reuse of models (found in the repository), users should be able to merge models. Multiple model support is a requirement for model merging, since access to the content of multiple models is needed for model integration. In the collaborative version of Garp3, multiple models can be opened. Each model is visualised as a tab. Clicking on a tab, or an editor associated to a model makes that model active. Having the ability to have multiple models open is required for the implementation of model merging support, as the content of different models is accessible at the same time. Multiple model support is described in Chapter 5.

With multiple model support in place, it is possible to implement model merging allowing reuse of models and model parts. This allows users to (1) easily base their modelling on existing work, (2) aggregate joint work into bigger models, and (3) create model variations that look at a phenomenon from a different perspective. The model merging functionality in Garp3 fulfils this requirement. The implementation of copy/paste functionality was added to allow users to copy a selection from one model into another model. This is a non-trivial issue, since a lot of model ingredients depend on each other in qualitative models. So definitions of pre-requisite model ingredients also have to be copied in order to maintain a consistent model. Model merging support is described in Chapter 6.

The Garp3 software and extensions facilitating collaborative work are available via the QRM portal (http://hcs.science.uva.nl/QRM/).

2. Sketch Environment

The Sketch module of the QRM workbench aims to support collaborative modelling by allowing users to represent initial and intermediate results during the modelling process, as described in [1, 8].

2.1. The Sketch Editors

The sketch environment is accessible from the Garp3 main screen, by pressing the ‘About this model and Sketch’ icon. On the top right of the resulting window, there will be seven icons to open the different Sketch editors. There are four different graphical editors in the Sketch environment, each belonging to a specific stage in the modelling process (shown within brackets):

1. Concept map editor (Orientation and initial specification)
2. Structural model editor (System selection and structural model)
3. Causal model editor (Global behaviour)
4. State-transition graph editor (Global behaviour)

In addition, there are three text-based editors, for the specification of processes, actions and external influences, and scenarios:

5. Process definitions editor (Global behaviour)
6. Actions and external influences editor (Global behaviour)
7. Scenario definitions editor (Global behaviour)

Each of the seven Sketch editors is presented in one of the following subsections.

2.2. The Concept Map Editor

A concept map consists of two primitives: nodes and arcs. Nodes reflect important concepts, while arcs show the relationships between those concepts [8]. The Concept Map Editor uses a visualization with text in different font styles for Concepts (Roman) and *Relations (Italics)*. An example screenshot of the Concept Map Editor is shown in Figure 1.

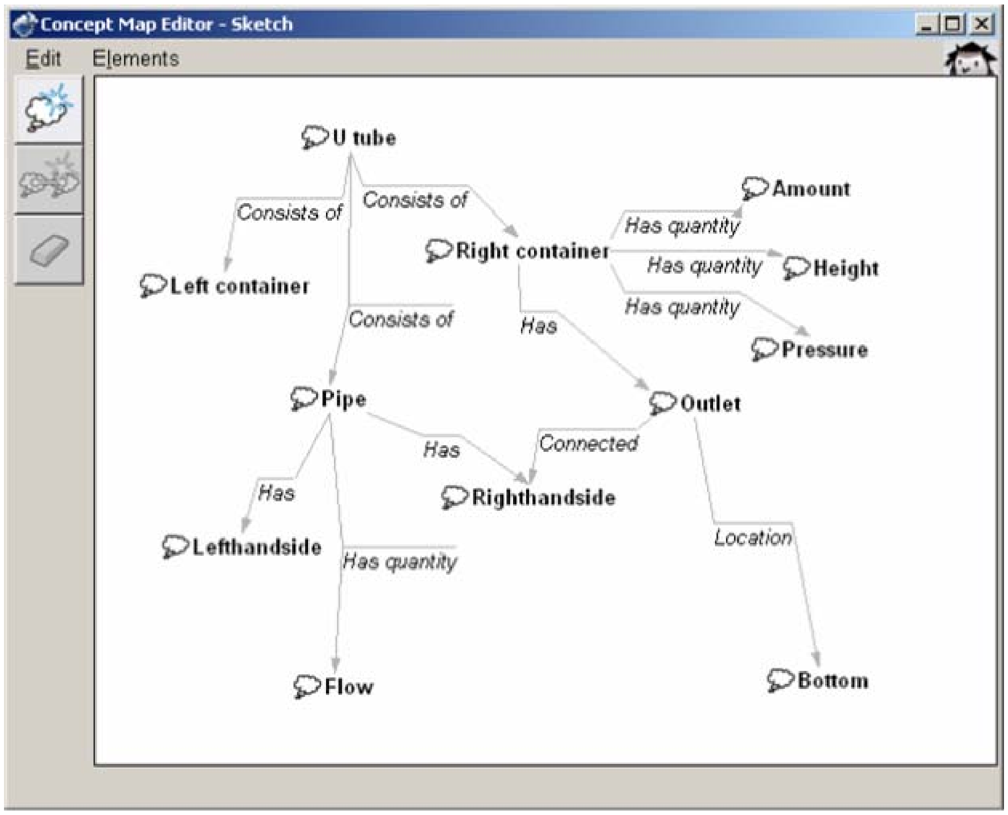


Figure 1: Concept Map Editor

Within the Concept Map Editor, the following user manipulations are possible:

- Add a concept or a relation[3]
- Select a concept or a relation (by mouse pointing and clicking)
- Select multiple concept nodes and/or relation arcs (by shift mouse pointing and clicking, or dragging a rectangle around them)
- Delete selected concept or relation[4]
- Edit text label for a concept or relation
- Add remarks for a concept or relation
- Freely move selected concept nodes and relation arcs in the graph
- Save graph into model (including the layout as created by the modeller)
- Save the diagram to an EPS file

2.3. The Structural Model Editor

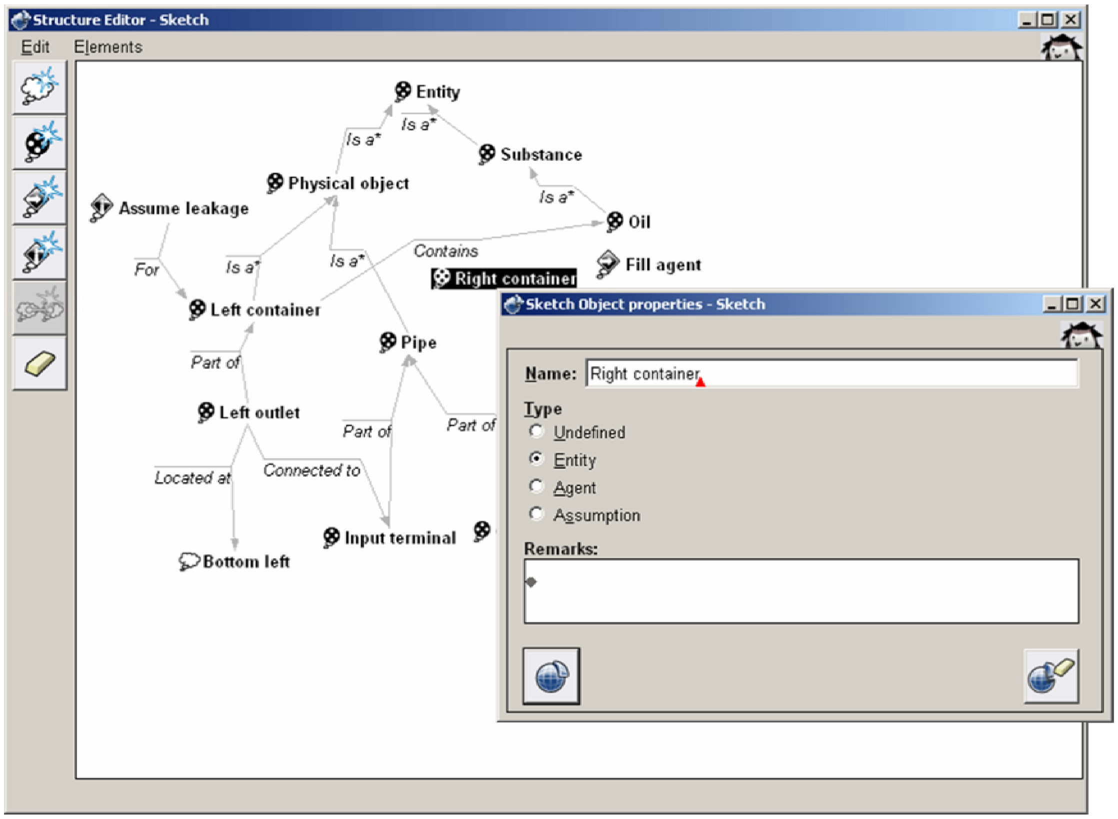
The Structural Model Editor is similar to the Concept Map Editor, but is tuned to specifying the structure of a system. Besides the addition of types of concepts, the Structural Model Editor differs from the Concept Map Editor in that it has predefined structural relations from which the user can choose when adding a relation. The list of predefined relations currently contains the following:

- Is a (subtype relationship)
- Contains
- Connected to

[3] When a relation is added, its definition will be added to a list of relations from which the user can choose when adding relations of the same kind later.
[4] When arc C connects node A and B, then delete arc C when node A or B (or both) is deleted.

- Part of

It is possible to add new relations as well. An example screenshot of the Structural Model Editor is shown in Figure 2.



Name: Right container

Type
- ( ) Undefined
- [✓ Entity]
- ( ) Agent
- ( ) Assumption

Remarks:

Figure 2: Structural Model Editor

Within the Structural Model Editor, the following user manipulations are possible:

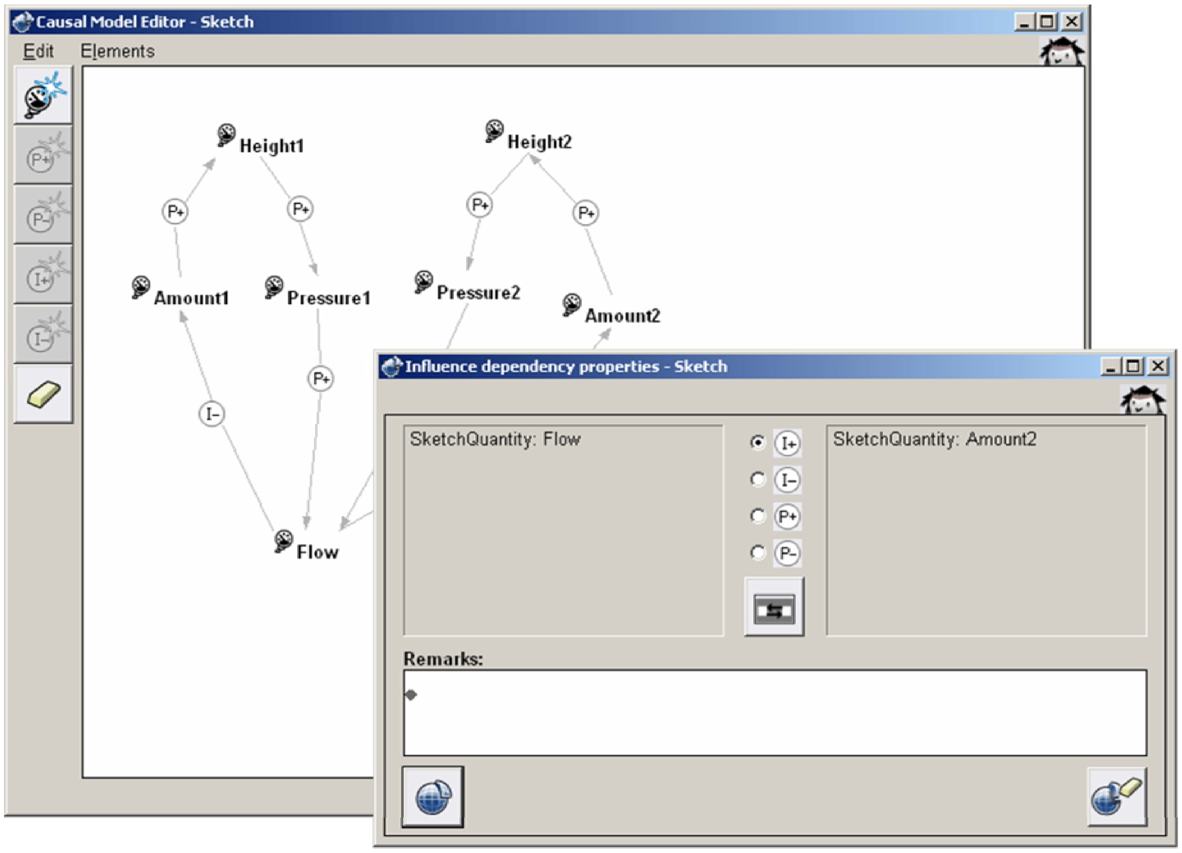
- Add a concept or a relation[5]
- Assign a type to a concept (Entity, Agent, Assumption, or Undefined–the default).
- Select a concept or a relation (by mouse pointing and clicking)
- Select multiple concept nodes and/or relation arcs (by shift mouse pointing and clicking, or dragging a rectangle around them)
- Delete selected concept or a relation[6]
- Edit text labels for each concept and relation
- Add remarks/comments to each concept and relation
- Freely move selected concept nodes and relation arcs in a graph
- Save graph into model (including the layout as created by the modeller)
- Save the diagram to an EPS file

[5] When a relation is added, its definition will be added to a list of relations from which the user can choose when adding relations of the same kind later.

[6] When arc C connects node A and B, then delete arc C when node A or B (or both) is deleted.

2.4. The Causal Model Editor

The Causal Model Editor allows the user to specify quantities and causal relationships between them. There are four types of causal dependencies: positive influence (I+), negative influence (I–), positive proportionality (P+), and negative proportionality (P–). An example screenshot from the Causal Model Editor is shown in Figure 3.



SketchQuantity: Flow

[✓ I+] I– P+ P–

SketchQuantity: Amount2

Remarks:

Figure 3: Causal Model Editor

Within the Causal Model Editor, the following user manipulations are possible:

- Add a quantity
- Add a positive or negative proportionality (P+, P-)
- Add a positive or negative influence (I+, I-)
- Modify the type of causal dependency (I+, I-, P+, P-)
- Select a quantity or a dependency (by mouse pointing and clicking)
- Select multiple quantity nodes and/or dependency arcs (by shift mouse pointing and clicking, or dragging a rectangle around them)
- Delete selected quantity or dependency[7]
- Edit text labels for each quantity and dependency
- Add remarks/comments to each quantity and dependency
- Freely move selected quantity nodes and dependency arcs in a graph
- Save graph into model (including the layout as created by the modeller)
- Save the diagram to an EPS file

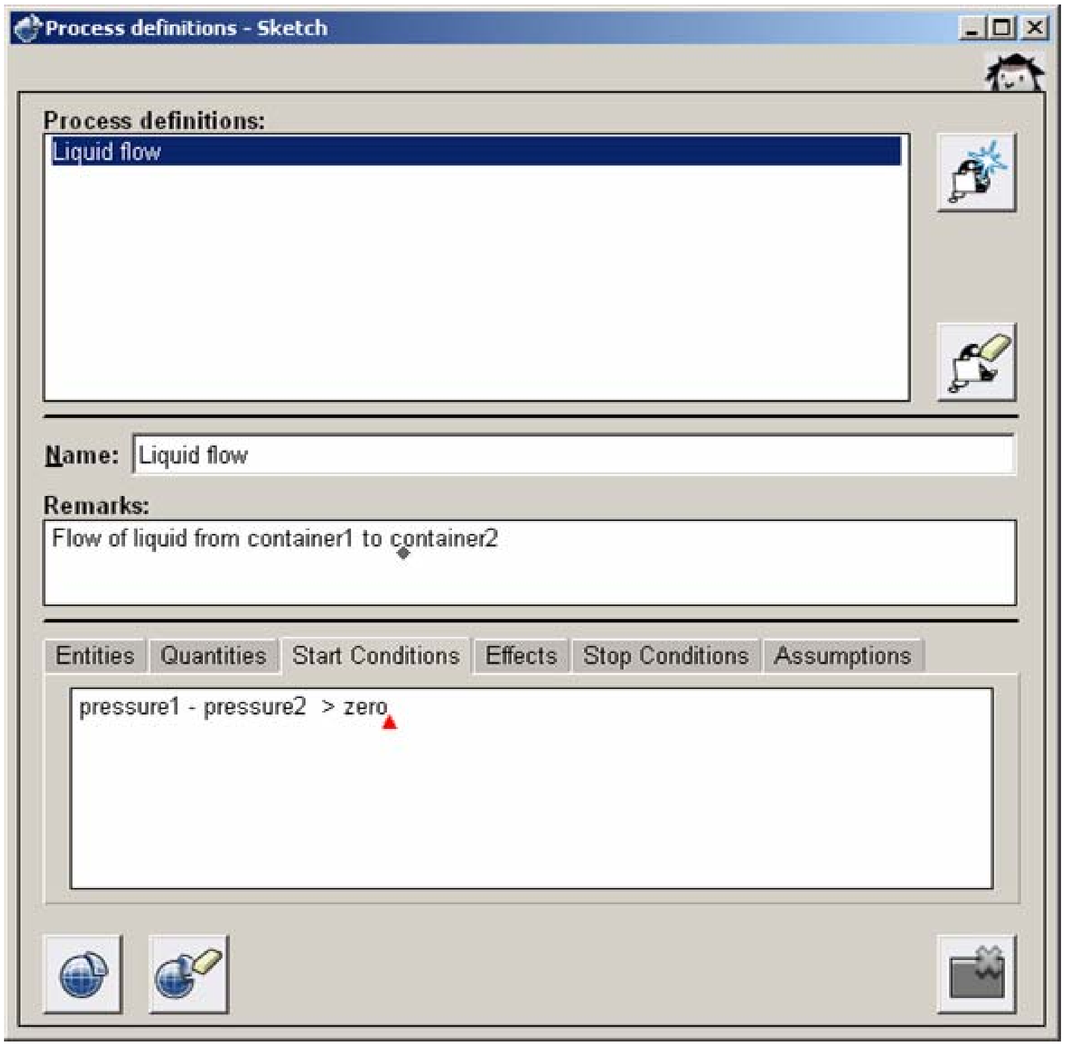
[7] When arc C connects node A and B, then delete arc C when node A or B (or both) is deleted.

2.5. The Process Definitions Editor

The Process Definitions Editor is a text-based editor which allows the user to specify relevant processes and their characteristics. An example screenshot of the Process Definitions Editor is shown in Figure 4.

Within the Process Definitions Editor, the following user manipulations are possible:

- Add a process definition
- Select a process definition
- Specify/modify the name of the process
- Specify/modify remarks about the process
- Specify/modify characteristics of a process, in the following text fields:
    - Entities
    - Quantities
    - Start conditions
    - Effects
    - Stop conditions
    - Assumptions
- Delete a process definition



Name: Liquid flow

Remarks: Flow of liquid from container1 to container2

pressure1 - pressure2 > zero

Figure 4: Process Definitions Editor

2.6. The Actions and External Influences Definitions Editor

The Actions and External Influences Editor is a text-based editor which allows the user to specify relevant actions and external influences and their characteristics, similar to the Process Definitions Editor. The main distinction is the addition of an agent that initiates the action. Within the Actions and External Influences Editor, the following user manipulations are possible:

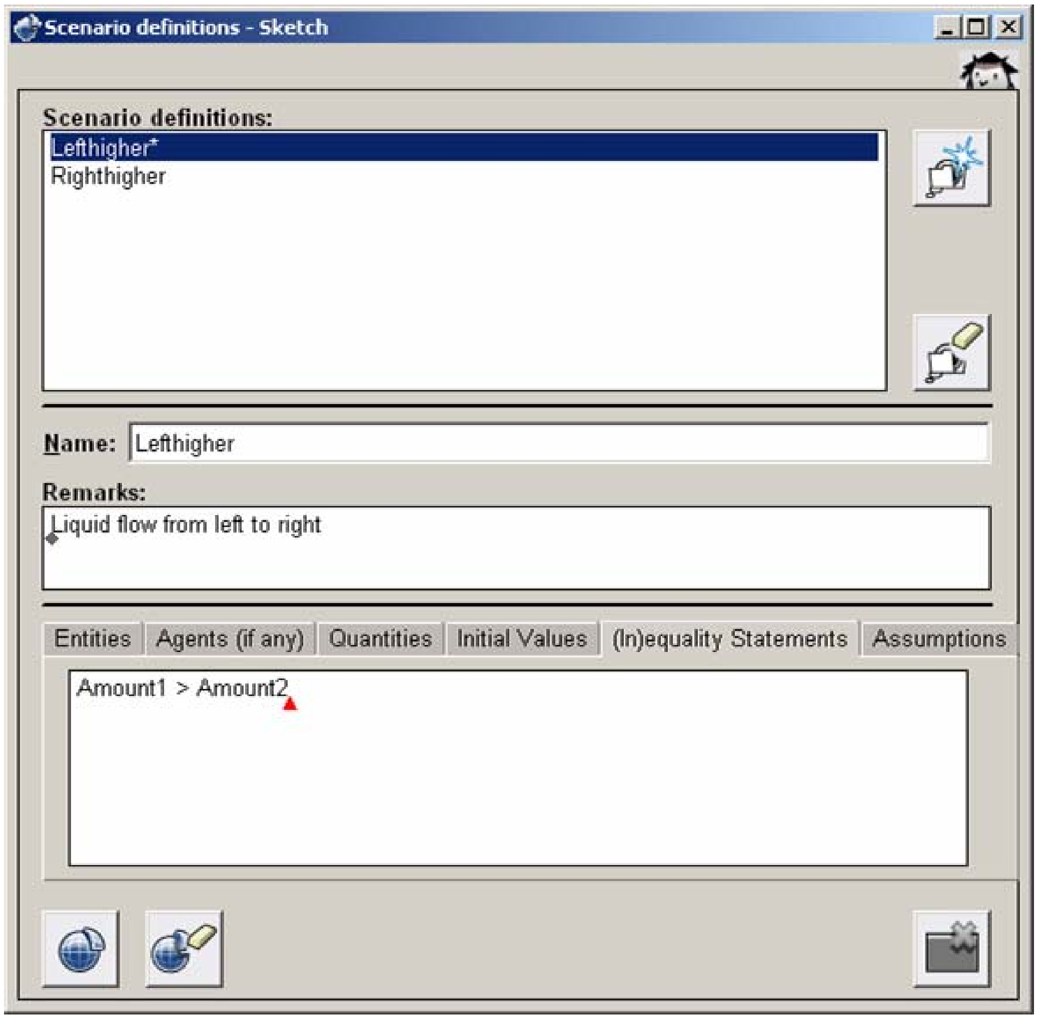
- Add an action/external influence definition
- Select an action/external influence definition
- Specify/modify the name of the action/external influence
- Specify/modify remarks about the action/external influence
- Specify/modify characteristics of an action/external influence, in the following text fields:
    - Agents
    - Entities
    - Quantities
    - Start conditions
    - Effects
    - Stop conditions
    - Assumptions
- Delete an action/external influence definition

2.7. The Scenario Definitions Editor

The Scenario Definitions Editor is a text-based editor which allows the user to specify scenarios and their characteristics, similar to the Process Definitions Editor and the Actions and External Influence Definitions Editor. An example screenshot of the Scenario Definitions Editor is shown in Figure 5.

Within the Scenario Definitions Editor, the following user manipulations are possible:

- Add a scenario definition
- Select a scenario definition
- Specify/modify the name of the scenario
- Specify/modify remarks about the scenario
- Specify/modify characteristics of a scenario definition, in the following text fields:
    - Entities
    - Agents (if any)
    - Quantities
    - Initial Values
    - (In)equality Statements
    - Assumptions
- Delete a scenario definition


Scenario definitions - Sketch

Scenario definitions:

Lefthigher*
Righthigher

Name: Lefthigher

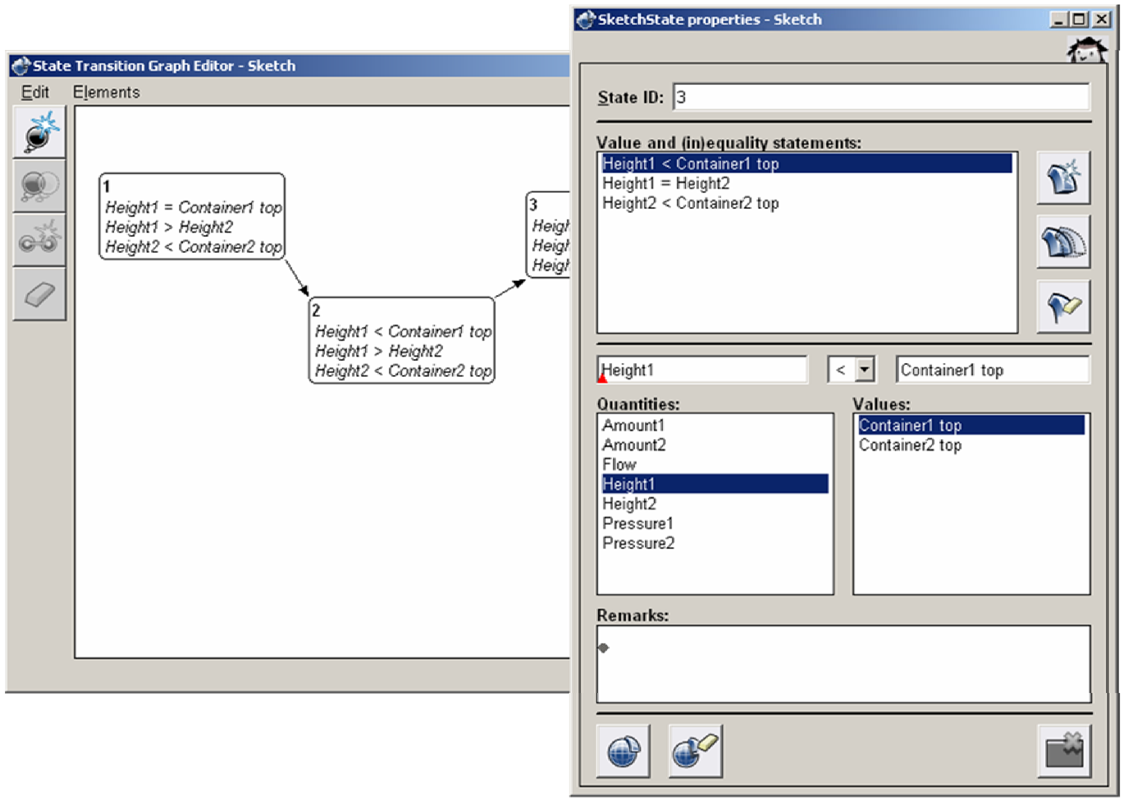
Remarks:
Liquid flow from left to right

Amount1 > Amount2

Figure 5: Scenario Definitions Editor

2.8. The State Transition Graph Editor

The State Transition Graph Editor supports specification of the typical states that the system exhibits, in terms of (in)equality statements between a quantity and a specific value, or between two quantities. A screenshot of the State Transition Graph Editor is shown in figure 6.


State Transition Graph Editor - Sketch
Edit
Elements
SketchState properties - Sketch
Value and (in)equality statements:
Quantities:
Values:
Remarks:

1
Height1 = Container1 top
Height1 > Height2
Height2 < Container2 top

2
Height1 < Container1 top
Height1 > Height2
Height2 < Container2 top

3

State ID: 3

Height1 < Container1 top
Height1 = Height2
Height2 < Container2 top

Height1 < Container1 top

Amount1
Amount2
Flow
Height1
Height2
Pressure1
Pressure2

Container1 top
Container2 top

Figure 6: State Transition Graph Editor

Within the State Transition Graph Editor, the following user manipulations are possible:

- Add a new state
- Add a transition from one state to another state
- Copy an existing state
- Delete a state or transition
- Select one or multiple state(s)
- Move one or multiple state(s) (the transitions will move accordingly automatically)
- Save graph into model (including the layout as created by the modeller)

While adding, or modifying a state, the following options are possible:

- Add a new inequality
    - To add an inequality between quantity and value, type a quantity name (or select a quantity) on the left hand side, and type a value name (or select a value) on the right hand side. Click save changes afterwards.
    - To add an inequality between two quantities, type a quantity name (or select a quantity) on the left hand side, and select another quantity on the right hand side, after activating the right hand side input field first. Click save changes afterwards.
- Select an inequality
- Copy an inequality
- Delete a selected inequality
- Type a name for a new quantity
- Modify the name for an existing quantity (first, select it, then type a new name)
- Type a name for a new value
- Modify the name for an existing value (first, select it, then type a new name)

- Modify the sign (>, >=, =, <=, <) of the inequality (default is ‘=’)
- Select a quantity or a value from the left or right list-box, respectively
- Add remarks/comments for the state
- Save the diagram to an EPS file

3. OWL Export and Import

To efficiently collaborate with other qualitative modellers, it should be possible to share models. This means models should be easily stored and retrieved from a central place. This functionality is provided to a modelling community via an online model repository (see Chapter 4).

One of the issues which had to be resolved was the file format for the exchange of qualitative models. The default format in which qualitative models are saved in the qualitative modelling and simulation workbench Garp3 [2] is binary. This makes it impossible to search for qualitative models in which model ingredient definitions with specific names occur, or in which primitives are arranged in a certain manner.

The Semantic Web initiative proposes that this issue can be resolved by making content machine accessible [3]. We have followed this proposal by formalising the QR vocabulary, i.e. the ingredients and their usage restrictions, in the Web Ontology Language (OWL) (see Figure 7). Based on this formalisation, called a generic ontology [5], an OWL format for qualitative models was developed [4]. These model formalisations can be considered domain ontologies [5], and can be opened in different OWL editors (see Figure 8). One of the research results of creating the formalisation is having shown that the formalisation of situations, and the reuse thereof, is a major unsolved issue for the OWL community [6].

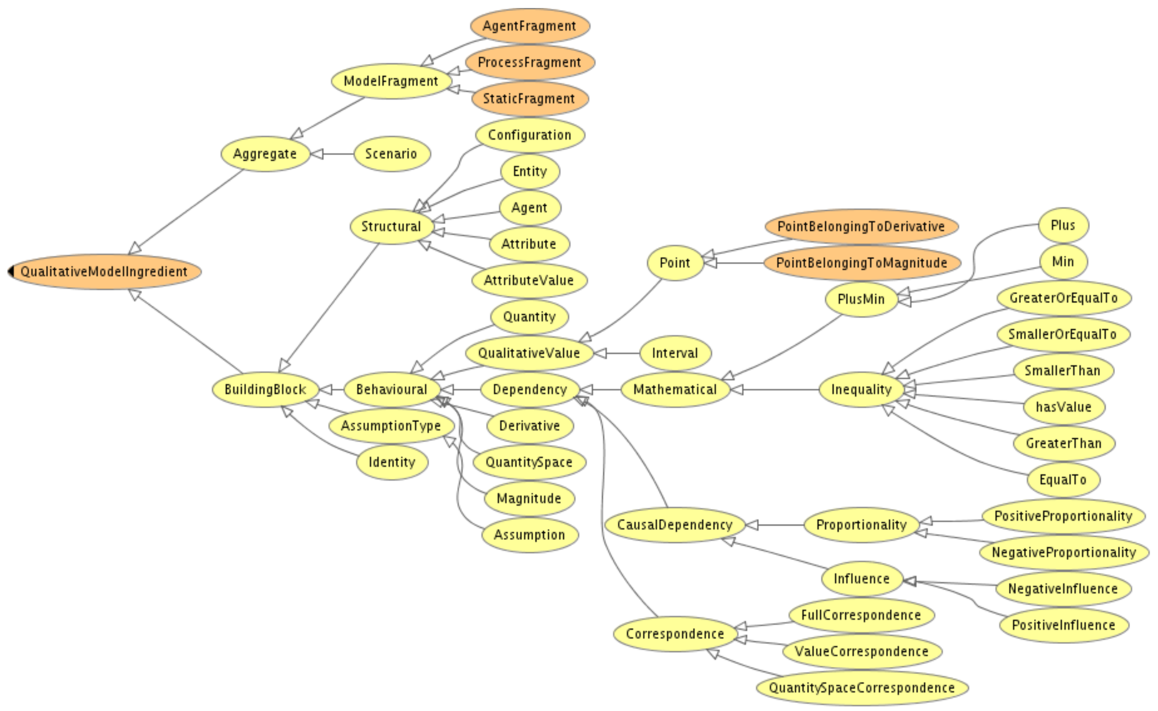


Figure 7: The ontology of qualitative modelling ingredients.

The development of a format is one the requirements for sharing and reuse of models. Additional functionality was implemented in Garp3 to export qualitative models to the OWL format and import them again. This chapter describes the issues that where encountered during the implementation of this functionality.

The models stored as OWL files can be uploaded and downloaded from a central online repository to allow easy interchange, searching and reuse of qualitative model content (see Chapter 4). The open format captures the semantics of the qualitative models, and allows OWL search engines to search inside models.

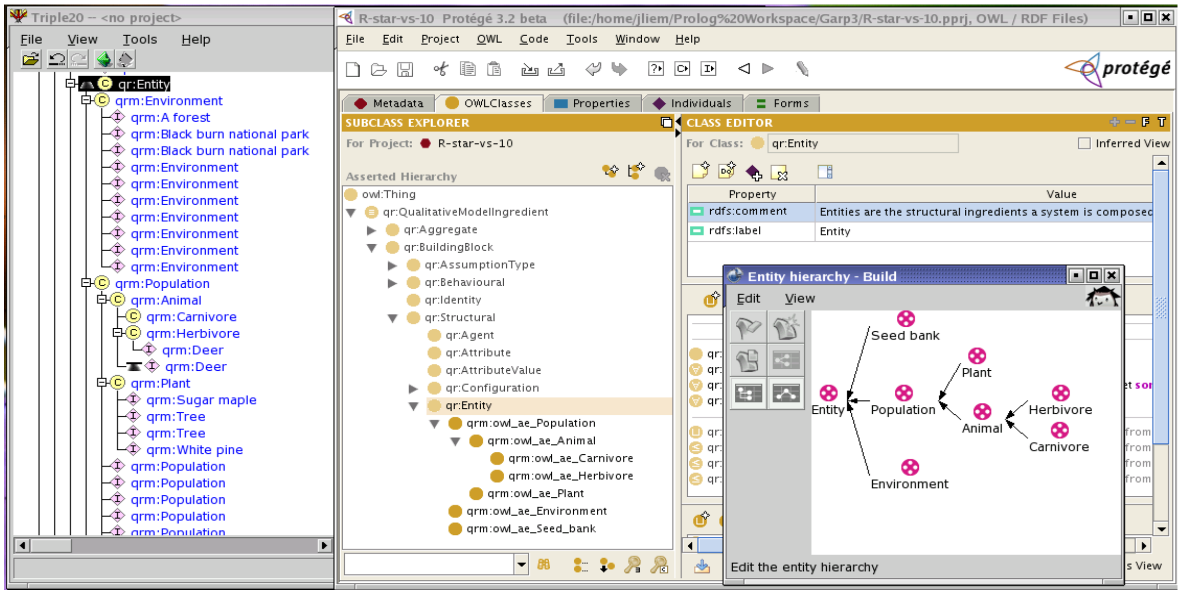


Figure 8: The R-Star model Entity Hierarchy in Triple 20 (left), Protege (middle to right) and Garp3.

3.1. Implementation

The main distinctions between the original Garp3 binary format and the representation of qualitative models in OWL are discussed in this section.

3.1.1. Reuse of Structures

The major difference between the OWL format and the internal Garp3 format is the reuse of model fragments. If a model fragment imports other model fragments in Garp3, the imported model fragments are actually references to the model fragment definitions. The layout information for these imported model fragments is stored in a separate data structure together with some identifying information. The advantage of this representation is that it is minimal in memory usage, although this should hardly be a consideration on today's hardware. The disadvantage is that the data structure becomes unnecessarily complex, since it is hard to distinguish between instances of the same imported model fragment in different contexts.

In the OWL format a more intuitive representation is chosen. Every imported model fragment is represented as an instance of the model fragment definition class. The model fragment that imports this model fragment has a *has_condition* relation to the imported model fragment instance. The content of the imported model fragment is represented as separate instances attached to this imported model fragment. This makes the file format more verbose, but there can be no uncertainty towards the model fragment to which the imported model fragment belongs.

3.1.2. Explicit Representation of Each Model Ingredient

In Garp3 not every element in the model is explicitly represented. For example the qualitative values of quantity spaces are encapsulated by the quantity spaces. The quantity spaces themselves are encapsulated by the quantities. The same quantity spaces and values are reused for each instance of the quantity. In the OWL format every element in the model is explicitly represented, which means each quantity space and each value is a separate instance. This explicit representation allows each model ingredient to be assigned its own layout, instead of storing it in a separate place.

3.1.3. Relations Between the Model Ingredients

Relations within model ingredients are not always represented as relations between those specific model ingredients in Garp3. For example, relations between quantity spaces or relations between values are represented as relations between quantities with some extra information. In the OWL format, because of the explicit representation of each model ingredient, the relations are truly between the model ingredients.

3.1.4. Unique Names of Model Ingredients

In Garp3, it does not matter if the same name is used multiple times for different objects of different types, although using the same name for a model ingredient of the same type is impossible. However, this causes problems in OWL, since each class or instance has to be identified by a unique URI. This was solved by adding prefixes to the URI's of the model ingredients for each type. For instances of model ingredients an integer is postfixed, which is incremented for each use of the model ingredient.

3.1.5. Correctness

The correctness and consistency of the model ontologies was checked using the Triple20 [7] and Protégé ontology editors and the Racer-Pro reasoner. Figure 2 shows part of a typical Garp3 model opened in Garp3, Protege and Triple 20. 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.

3.2. Functionality and Integration

The OWL functionality has been carefully integrated in file operations in Garp3. Users can choose to save their model to an OWL file, instead of the default HGP file. The same is true for loading files; it is possible to select an OWL file instead of a HGP file (see Figure 9).

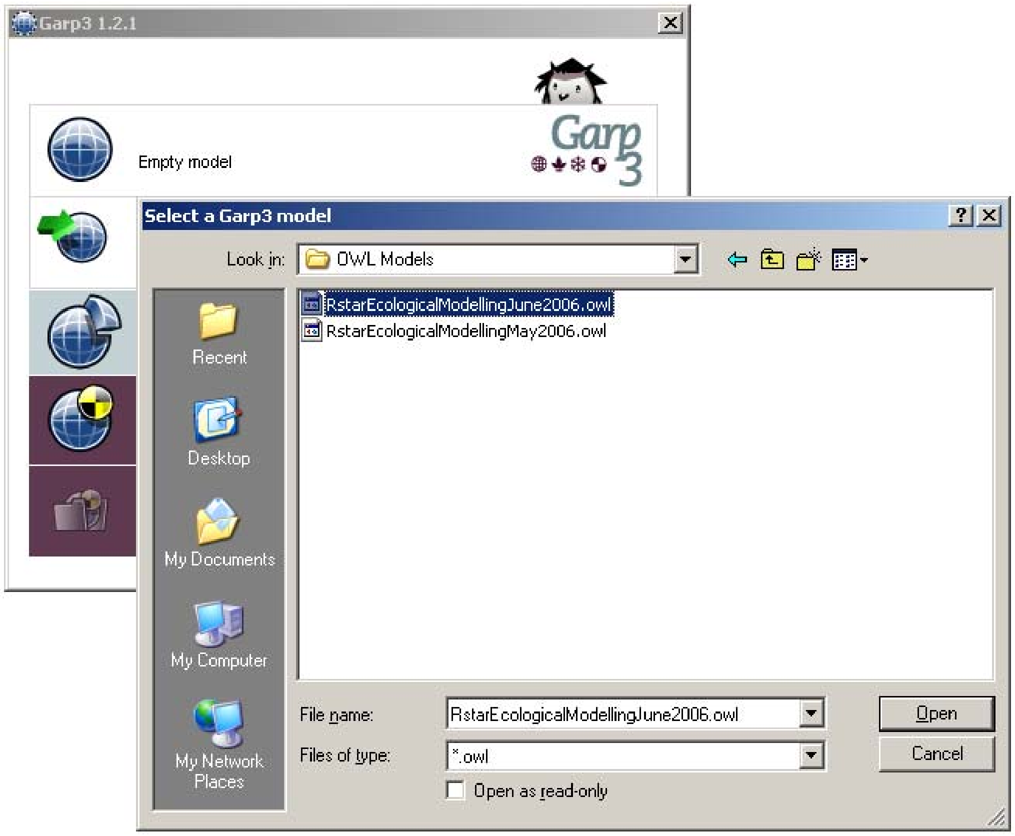


Figure 9: Loading an OWL file

In addition to the file format to which the model is saved, there is another difference in the loading and saving routines. The OWL functionality exports a qualitative model to a file, but some of the information is not exported, such as model simulation results and Sketches. Therefore, we like users to keep using HGP files to work with, and only use OWL files when they want to share their results. For this reason exporting a file to OWL keeps the current HGP file active (i.e. when pressing save again, the model is saved to a HGP file again). When loading an OWL file, the loaded model is considered a new changed model, which means the user is asked to save his file when he closes the model. This prevents users of losing work when working with Garp3.

Depending on user feedback we will consider adding model Sketches to the OWL format. Model results will not be saved, as it would make the model file impractically large. Moreover, they can simply be obtained by running the model.

4. Model Repository

The collaborative version of Garp3 allows users to export models to an OWL format, and import them again (see Chapter 3). The OWL qualitative model files can be stored in the qualitative model repository[8], which will be integrated with the QRM Portal in due course[9]. This chapter describes the model repository. More details can be found in D2.3.2 [11].

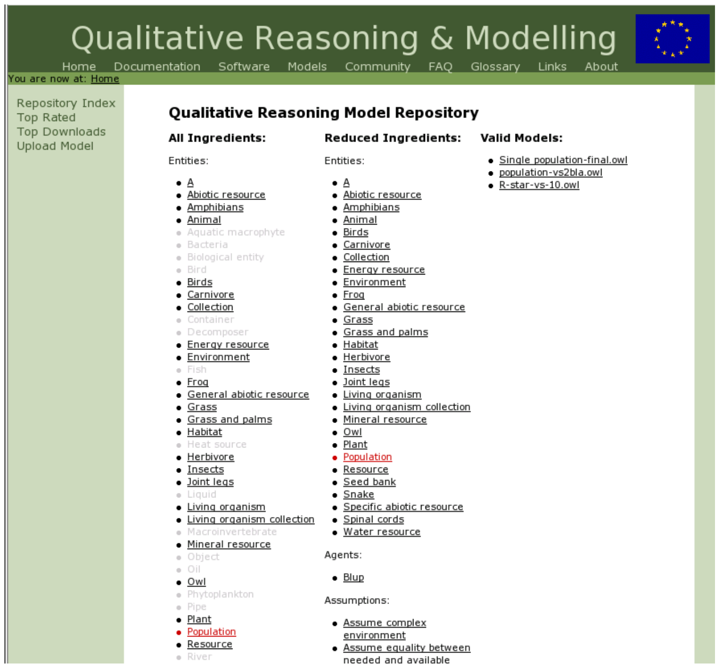
4.1. Model Repository Index

The main page of the repository is used for model search (see Figure 10 and 11), and shows the model ingredients in all models in the repository (‘All Ingredients’), the model ingredients in matching models (‘Reduced Ingredients’), and the models that contain the selected model ingredients (‘Valid Models’). The selected model ingredients are shown in red, while the model ingredients that do not match the query are shown in grey. These non-occurring model ingredients cannot be selected, as no models would match the query if the ingredient were added. Model ingredients can be removed by from a query by deselecting them. Furthermore, the model ingredient types can be pressed to collapse the list of model ingredients of that type. This allows faster navigation of the model ingredient types.

Selecting an unselected model ingredient from ‘All Ingredients’, or ‘Reduced Ingredients’ reduces the amount of matching models to the set that contains all the selected model ingredients. If multiple ingredients are selected, the repository searches for models in which both model ingredients occur. This allows users to iteratively refine their search for interesting models. Figure 10 shows the models that contain an entity ‘Population’, while Figure 11 shows a refined search for models that contain the entities ‘Population’, ‘Carnivore’ and ‘Plant’. Note that entities are not the only model ingredients that can be searched for. All model ingredient definitions in the models can be used in a query.

Clicking on one of the matching models brings up further information about the model (see Figure 12). The model information is distilled from the OWL file and contains the title, author, contributors, keywords, model goals, target audience, model version, model limitations, language, bibliographic citation, abstract and remarks. Each model also has a rating and a number of downloads. This information is stored separately from the model in a relational database. If the user has not submitted a rating already, it is possible to submit a rating to indicate the quality of the model. Finally, there is a link that allows the model to be downloaded.

[8] http://www.science.uva.nl/~jliem/Repository/
[9] http://hcs.science.uva.nl/QRM/


Qualitative Reasoning & Modelling
Home Documentation Software Models Community FAQ Glossary Links About
You are now at: Home
Repository Index
Top Rated
Top Downloads
Upload Model

Qualitative Reasoning Model Repository

All Ingredients:

Entities:

- A
- Abiotic resource
- Amphibians
- Animal
- Aquatic macrophyte
- Bacteria
- Biological entity
- Bird
- Birds
- Carnivore
- Collection
- Container
- Decomposer
- Energy resource
- Environment
- Fish
- Frog
- General abiotic resource
- Grass
- Grass and palms
- Habitat
- Heat source
- Herbivore
- Insects
- Joint legs
- Liquid
- Living organism
- Living organism collection
- Macroinvertebrate
- Mineral resource
- Object
- Oil
- Owl
- Phytoplankton
- Pipe
- Plant
- Population
- Resource
- River

Reduced Ingredients:

Entities:

- A
- Abiotic resource
- Amphibians
- Animal
- Birds
- Carnivore
- Collection
- Energy resource
- Environment
- Frog
- General abiotic resource
- Grass
- Grass and palms
- Habitat
- Herbivore
- Insects
- Joint legs
- Living organism
- Living organism collection
- Mineral resource
- Owl
- Plant
- Population
- Resource
- Seed bank
- Snake
- Specific abiotic resource
- Spinal cords
- Water resource

Agents:

- Blup

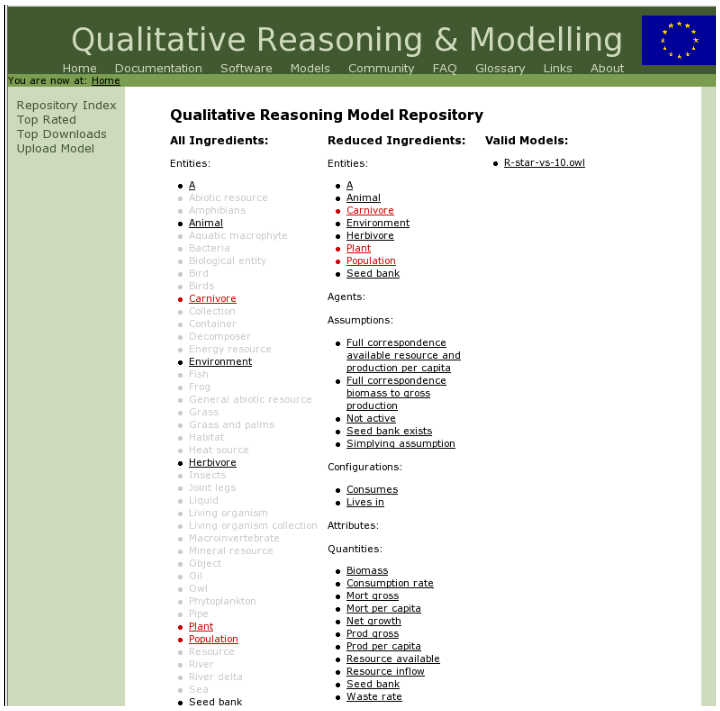
Assumptions:

- Assume complex environment
- Assume equality between needed and available

Valid Models:

- Single population-final.owl
- population-vs2bla.owl
- R-star-vs-10.owl

Figure 10: The model repository showing the models that contain the entity ‘Population’, and model ingredients in those models.


Qualitative Reasoning & Modelling
Home Documentation Software Models Community FAQ Glossary Links About
You are now at: Home
Repository Index
Top Rated
Top Downloads
Upload Model

Qualitative Reasoning Model Repository

All Ingredients:

Entities:

- A
- Abiotic resource
- Amphibians
- Animal
- Aquatic macrophyte
- Bacteria
- Biological entity
- Bird
- Birds
- Carnivore
- Collection
- Container
- Decomposer
- Energy resource
- Environment
- Fish
- Frog
- General abiotic resource
- Grass
- Grass and palms
- Habitat
- Heat source
- Herbivore
- Insects
- Joint legs
- Liquid
- Living organism
- Living organism collection
- Macroinvertebrate
- Mineral resource
- Object
- Oil
- Owl
- Phytoplankton
- Pipe
- Plant
- Population
- Resource
- River
- River delta
- Sea
- Seed bank

Reduced Ingredients:

Entities:

- A
- Animal
- Carnivore
- Environment
- Herbivore
- Plant
- Population
- Seed bank

Agents:

Assumptions:

- Full correspondence available resource and production per capita
- Full correspondence biomass to gross production
- Not active
- Seed bank exists
- Simplying assumption

Configurations:

- Consumes
- Lives in

Attributes:

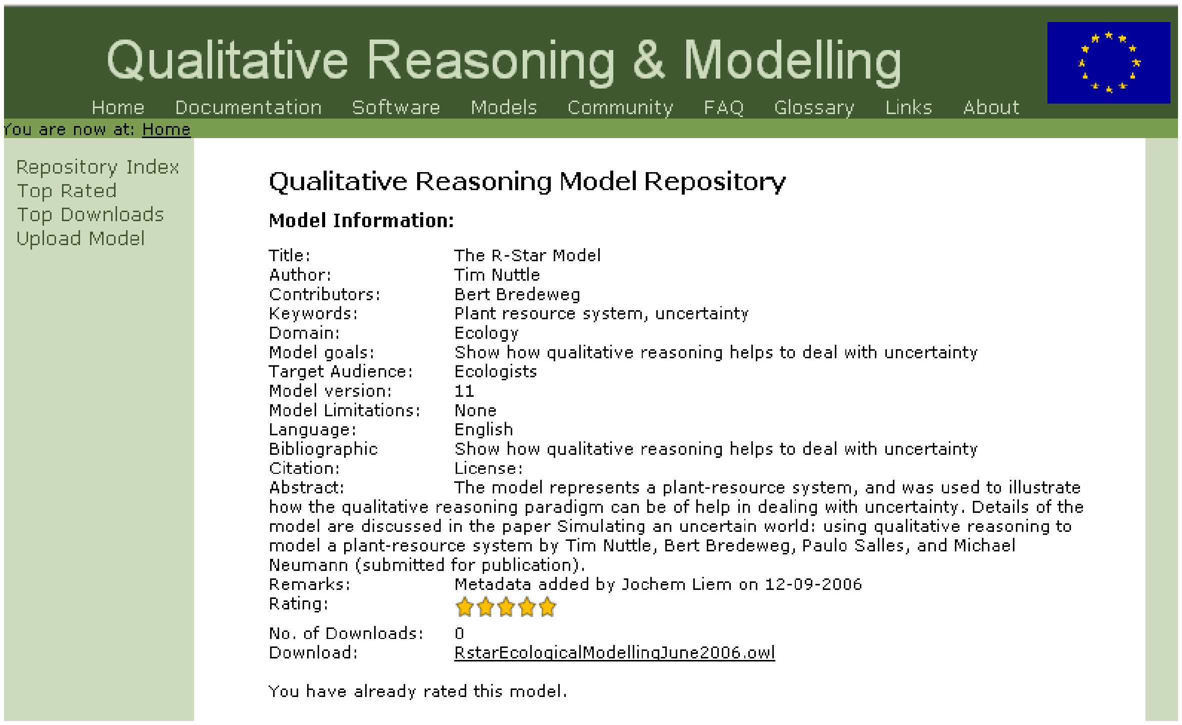
Quantities:

- Biomass
- Consumption rate
- Mort gross
- Mort per capita
- Net growth
- Prod gross
- Prod per capita
- Resource available
- Resource inflow
- Seed bank
- Waste rate

Valid Models:

- R-star-vs-10.owl

Figure 11: The model repository showing the models which contain ‘Population’, ‘Plant’, and ‘Carnivore’, and model ingredients in those models.


Qualitative Reasoning & Modelling
Home Documentation Software Models Community FAQ Glossary Links About
You are now at: Home
Repository Index
Top Rated
Top Downloads
Upload Model

Qualitative Reasoning Model Repository

Model Information:

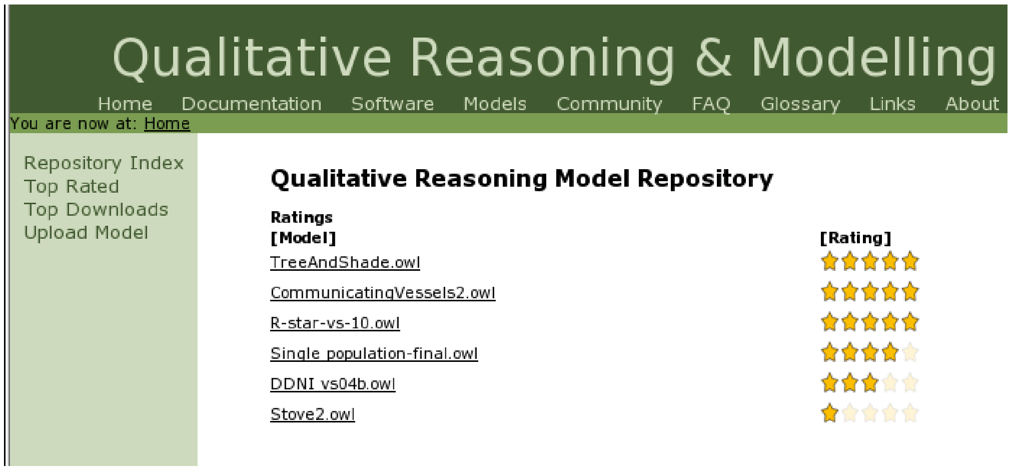
Title:	The R-Star Model
Author:	Tim Nuttle
Contributors:	Bert Bredeweg
Keywords:	Plant resource system, uncertainty
Domain:	Ecology
Model goals:	Show how qualitative reasoning helps to deal with uncertainty
Target Audience:	Ecologists
Model version:	11
Model Limitations:	None
Language:	English
Bibliographic	Show how qualitative reasoning helps to deal with uncertainty
Citation:	License:
Abstract:	The model represents a plant-resource system, and was used to illustrate how the qualitative reasoning paradigm can be of help in dealing with uncertainty. Details of the model are discussed in the paper Simulating an uncertain world: using qualitative reasoning to model a plant-resource system by Tim Nuttle, Bert Bredeweg, Paulo Salles, and Michael Neumann (submitted for publication).
Remarks:	Metadata added by Jochem Liem on 12-09-2006
Rating:	★★★★★
No. of Downloads:	0
Download:	RstarEcologicalModellingJune2006.owl

You have already rated this model.

Figure 12: The repository showing specific information about a qualitative model, and the possibility to download it.

4.2. Top Ratings

The top rated link on the left side brings the user to a page which shows a list of models ordered by their rating (see Figure 13). The ratings are actually a real number between 0 and 5. Therefore the ordering of models with the same amount of stars reflect the more precise rating in the database. Note that the ratings in Figure 13 are for demonstrational purposes and do not reflect the actual quality of the models.


Qualitative Reasoning & Modelling
Home Documentation Software Models Community FAQ Glossary Links About
You are now at: Home
Repository Index
Top Rated
Top Downloads
Upload Model

Qualitative Reasoning Model Repository

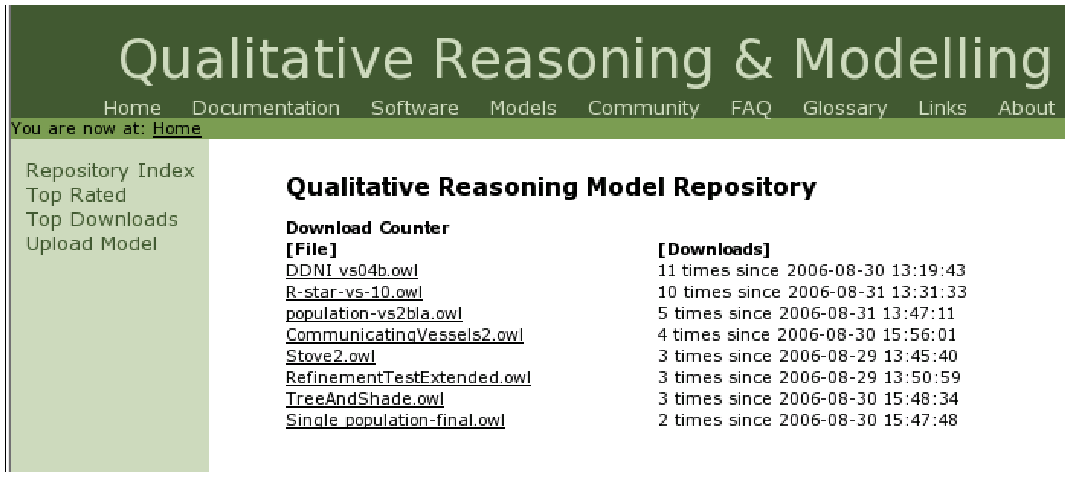
Ratings

[Model]	**[Rating]**
TreeAndShade.owl	★★★★★
CommunicatingVessels2.owl	★★★★★
R-star-vs-10.owl	★★★★★
Single population-final.owl	★★★★☆
DDNI vs04b.owl	★★★☆☆
Stove2.owl	★☆☆☆☆

Figure 13: The ‘Top Rated’ page shows the list of the models ordered by their rating (fictional data).

4.3. Top Downloads

The top downloads link brings the user to a page which shows a list of models ordered by the number of times they are downloaded (see Figure 14). The page also shows the data on which the model was first downloaded. Clicking on a model bring the user to a page showing more information about the model, and the possibility to download it (see Figure 14). Note that the number of downloads shown in Figure 14 have been manipulated to show the functionality, and do not reflect the actual popularity of the models.



Qualitative Reasoning Model Repository

Download Counter

[File]	[Downloads]
DDNI vs04b.owl	11 times since 2006-08-30 13:19:43
R-star-vs-10.owl	10 times since 2006-08-31 13:31:33
population-vs2bla.owl	5 times since 2006-08-31 13:47:11
CommunicatingVessels2.owl	4 times since 2006-08-30 15:56:01
Stove2.owl	3 times since 2006-08-29 13:45:40
RefinementTestExtended.owl	3 times since 2006-08-29 13:50:59
TreeAndShade.owl	3 times since 2006-08-30 15:48:34
Single population-final.owl	2 times since 2006-08-30 15:47:48

Figure 14: The ‘Top Downloads’ page showing a list of the models that are downloaded the most (functional data).

4.4. Upload Model

The ‘Upload Model’ page allows users to upload their qualitative models (see Figure 15). Clicking the ‘Browse…’ button brings up a file manager in which the user can select a model file. Pressing ‘Upload Model’ uploads the model to the repository. The repository checks if the model is a valid XML file that ends with the .owl extension. If the model is not a valid .owl file, the repository indicates this and the user can try to upload a different model.

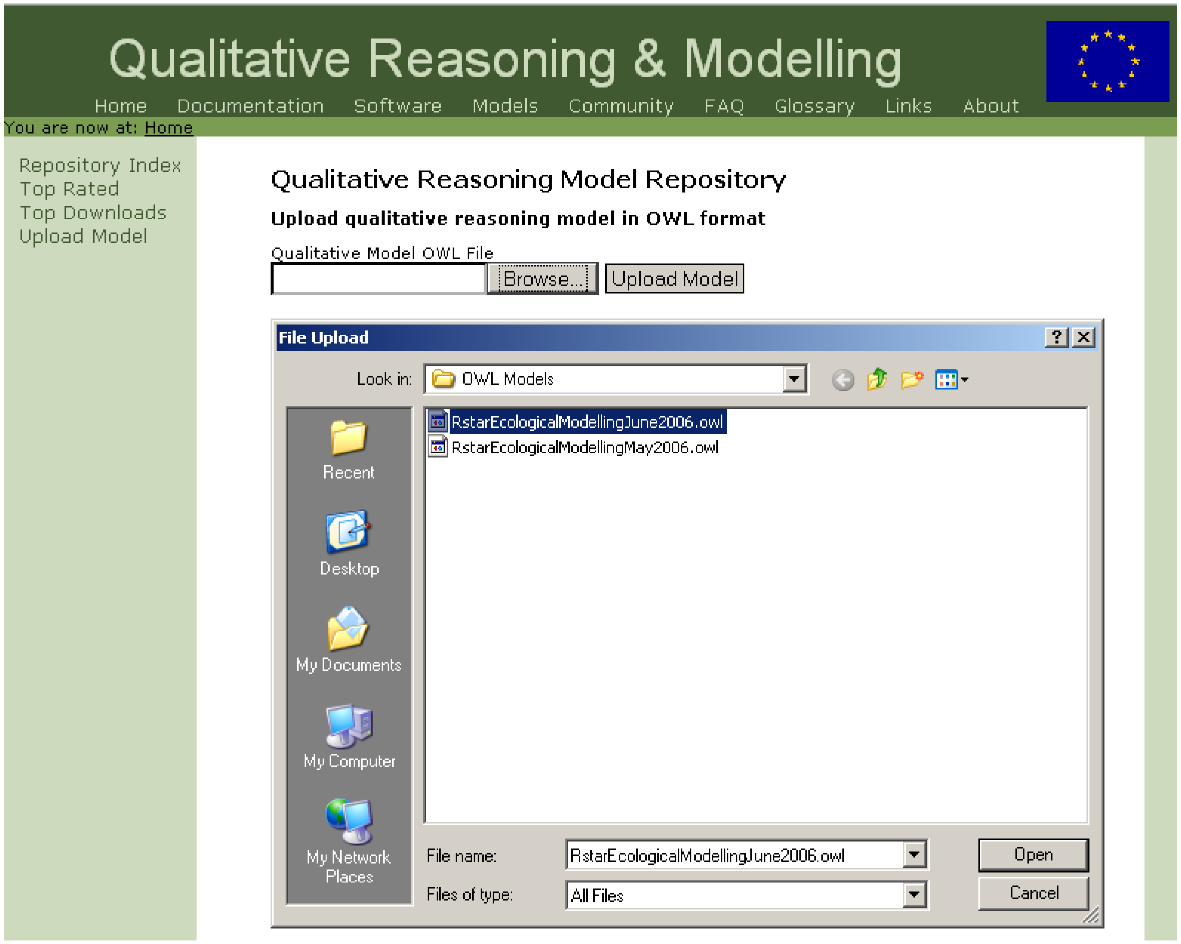

Qualitative Reasoning & Modelling
Home Documentation Software Models Community FAQ Glossary Links About
You are now at: Home
Repository Index
Top Rated
Top Downloads
Upload Model
Qualitative Reasoning Model Repository
Upload qualitative reasoning model in OWL format
Qualitative Model OWL File
Browse...
Upload Model
File Upload
Look in: OWL Models
Recent
Desktop
My Documents
My Computer
My Network Places
RstarEcologicalModellingJune2006.owl
RstarEcologicalModellingMay2006.owl
File name: RstarEcologicalModellingJune2006.owl
Files of type: All Files
Open
Cancel

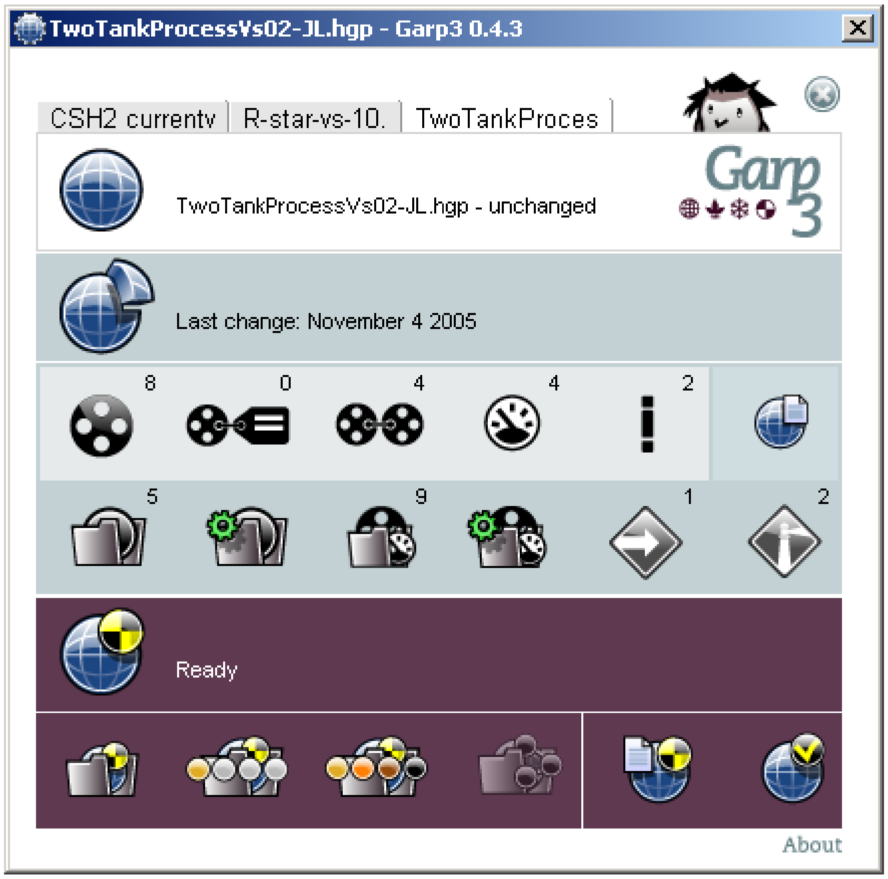
Figure 15: The upload page, and the file browser in which a file can be selected.

5. Multiple Model Support

The OWL functionality and qualitative model repository have made it possible to share models between modellers. However, to allow reuse an additional model-merging functionality is required (see Chapter 6), which is provided in the form of copy/paste functionality. To be able copy/paste between different models, and model parts, it is necessary have multiple models open simultaneously.

Multiple model support has been added to the collaborative version of Garp3. Each open model is visualised as a tab (see Figure 16). The program opens with a single tab with a blank model called model1. Opening models either opens the file in a blank tab (if the model is blank), or opens a new tab to load the model in. Clicking a tab makes that model active (possible to edit and simulate). Due to legacy code in the simulate environment it is not possible to run multiple simulations at the same time. Running a simulation in a different tab closes any currently existing simulation.

If multiple editors from different models are open, clicking on an editor makes the model that corresponds to that editor active. Users should take care to notice that the active model is changed without having to click in the main screen.



TwoTankProcessVs02-JL.hgp - unchanged

Last change: November 4 2005

Ready

Figure 16: Garp3 with multiple model support. Three models are open.

5.1. Implementation

5.1.1. The Global @model Variable

Garp3 was not designed with support for multiple models in mind. As a result, part of the architecture had to be changed to add this functionality. The major problem is that in the single-user version of Garp3, the model is referred to using a global variable called @model. This global variable is used throughout the code base.

Due to the frequent occurrences of the @model variable, it would have taken to much time to add an extra model variable to all the function definitions in Garp3. As an alternative solution we have changed the @model variable to become an instance of the *var* class, which is a sort of pointer that points to an object. Calls to the @model *var* object are automatically redirected to the object it refers to.

When another model is made active, the @model pointer is redirected to the new model.

5.1.2. Tabs as Buttons

The tabs as found in SWI-Prolog/XPCE are not suitable to be used as buttons, as no API is provided to treat them as such. As a result, our own implementation of buttons had to be developed based on the existing tab classes. This was done by creating a subclass of the existing tab class, and adding event-handling code to make them act like buttons.

5.1.3. Resizable Tabs

Tabs seemed to be the natural way to represent open models in Garp3. Unfortunately, the size of model names are often long, which causes the tabs to run of the Garp3 screen when multiple models are open. To solve this issue the tabs are automatically resized, by dividing the total available space by the number of open tabs. Unfortunately, resizing of tabs was not natively possible in SWI-Prolog/XPCE. Since there was no time to implement this functionality in the programming language, the number of characters is the tabs were changed depending on the number of open models. If more models are opened, the number of characters in the tabs is reduced. This works good enough for our purposes, but does not scale, as each tab has margins on the left and right sides. Therefore, if enough models are opened, it is still possible for the tabs to appear (partially) outside the Garp3 interface. We have fixed the maximum number of models that can be opened to 4. This fixes the issue, and has the added benefit of preventing users to have to many models open.

5.1.4. The Correct Active Models

As a result of the global @model variable, special care had to be taken to make sure the correct model was active at the right time. Clicking on the correct tab to make models active was not enough, as creating a new definition or instance in an editor could create that definition or instance in the wrong model, potentially crashing Garp3. To solve this issue, the model associated with an editor has to be made active once an editor has been made active. This is achieved by keeping track of which models are associated to which windows. The click events on windows are captured, and cause the associated model to become active.

5.1.5. Closing Windows

When a model is closed, the associated windows have to be closed, as using those editors without the associated model could cause crashes. To have the correct model active at all times we save the association between editor windows and models. When a model is closed, the windows associated to this model are simply closed too.

6. Model Merging

Model merging functionality makes it possible for users to (1) easily base their modelling work on existing work, (2) aggregate joint work into bigger models, and (3) create model variations that look at a phenomenon from a different perspective. To facilitate this creation of new models, models can be searched for and downloaded from the model repository, and imported in Garp3, and merged with other models using the merging functionality (copy/paste).

6.1. Entity, Agent and Assumption Definitions

Entities, agents and assumptions have no extra semantics next to their name. Therefore, the model merging functionality should not create a new model ingredient definition if the target model already has a definition with the same name. The only drawback is that the definition in the target model could have different 'remarks' compared to the definition in the source model. This difference is ignored by the algorithm. An alternative would be to create a new definition even if the model ingredient definition already exists (e.g. By appending “(copy)” to the name). Since the models from which parts are merged are probably from the same domain, definitions with the same name likely refer to the same concept. Furthermore, an entity with 'copy' postfix would not have explanatory power in a model, and would have to be renamed anyway (the remarks would have to be changed too). Users could instead just have created this new concept. For these reasons copy/pasting a definition to a target model which already has this definition has no effect.

Copying a set of entities, agents or assumptions can be done by selecting them and pressing copy (Edit=>Copy, or CTRL+C)[10]. This action removes the current entities, agents or assumptions in the copy buffer (which is a model itself), and adds the selected model ingredients and their parents to the copy buffer. The parents are added so that the information about the parents of the selected model ingredients is stored in a natural way. Pasting a set of entities, agents or assumption is done by optionally selecting an existing definition and pressing paste (Edit=>Paste, or CTRL+V). The set of copied definitions is integrated in the target model in order of depth. If the parent of the definition already exists, the definition is created underneath it, even if another root is selected. The last two choices ensure that a sub-hierarchy in the source model is preserved in the target model. If no selection and no parent are available the copied definitions are pasted below the root node (Entity, Agent or Assumption).

There is one exception in which case it is essential to use a suffix behind definition names. Having the same name in multiple hierarchies is illegal. Therefore pasting a definition to a model that has a definition with the same name in a different hierarchy needs some intervention. This is solved by adding the postfix “(also *type)”* behind the pasted definition name, where *type* is the type of the definition with the same name which is already in the model. So adding an agent 'Fire' to a model which already has an entity 'Fire' would result in the agent being renamed 'Fire (also entity)'.

[10] On MacOSX, users can also use Apple+C to copy, and Apple+P to paste.

6.2. Configuration Definitions

Configurations, similarly to entities, agents and assumptions, have no extra semantics other than their name. Therefore, they are treated in the same way, i.e. when pasted they are ignored if the name already exists in the target model. Adding them with a postfix '(copied)' is not useful, as they would have to be redefined anyway, which is more work than just creating a new definition. Furthermore, a user could have missed the definition in the target model. In contrast to entities, agents and assumptions, configurations are not organised in a hierarchy.

Currently, it is only possible to have a single selection at the same time in the configuration definition editor. Therefore, configuration definitions can only be copied to another model one at a time. Copying is done by selecting a configuration and pressing CTRL+C, which removes the configurations in the copy buffer, and adds the configuration definition to it. Pasting is done by pressing CTRL+V, and creates the configuration in the target model.

6.3. Attribute Definitions

In contrast to the previous model ingredient definitions, attributes have more semantics than just their name. The possible values of attributes, called attribute values, are essential to the definition of an attribute. Similar like in the configuration editor, it is only possible to select a single attribute at a time. This selection can be copied by pressing CTRL+C, which deletes the existing attribute definitions in the copy buffer, and copies the selected attribute definition to the copy buffer. Attribute definitions can be pasted to another model by pressing CTRL+V.

When pasting attributes into a model, the program should take the attribute values in consideration. If an attribute already exists which has both the same name, and the same values, then it is ignored. Creating exactly the same attribute and adding the postfix ‘(copy)’ is not useful, as it would have to be edited to create a usable attribute definition. Furthermore, if a user really does want such a copy, the clone button[11] can be used instead. In conditions when the name is equal, but the values are not, the attribute is added with the postfix ‘(different values)’. Of both the name and the values are different, the attribute definition is created normally.

6.4. Quantity Space Definitions

Quantity space definitions have the same data structure as attributes, i.e. a name and a set of values. The editor only allows the selection of a single quantity space at a time. This selection can be copied by pressing CTRL+C, which deletes the existing quantity space definitions in the copy buffer, and copies the selected quantity space definition to the copy buffer. Quantity space definitions can be pasted to another model by pressing CTRL+V.

The only difference with attributes is that unlike attributes values the quantity space values describe an ordered set of values. Therefore, pasting a quantity space is only ignored if a quantity space already exists which has both same the name, the same set of values, and the same order of the values. If the name is equal, but either the set of

[11] The copy button available in all Build workspaces in the Garp3 workbench.

values or the order of these values differs, a new quantity space definition is created with the postfix '(different values)'.

6.5. Quantity Definitions

Quantity definitions are defined by a name and a set of allowed quantity spaces. The editor only allows the selection of a single quantity at a time. This selection can be copied by pressing CTRL+C, which deletes the existing quantity definitions in the copy buffer, and copies the selected quantity definition to the copy buffer. Quantity definitions can be pasted to another model by pressing CTRL+V.

In contrast to the other definitions, it is possible to merge two quantities when a quantity is copied to another model with a quantity with the same name. Again, if both the name and the set of possible quantity spaces are equal, there is no reason to create a new quantity definition. If a user wants to create a similar quantity definition, the clone button can be used. If the names are equal, but the possible quantity spaces are not, the quantity spaces in the quantity definition from the source model can be added to the list of quantity spaces in the quantity space definition of the target model.

6.6. Model Fragment Definitions

Model fragments are composed out of instances of model ingredients (which can include instances of model fragments). As a result, comparing model fragments with each other is rather complex. A set of model fragments can be selected in the model fragment canvas. The selection can be copied by pressing CTRL+C, which deletes the existing model fragment definitions in the copy buffer, and copies the selected model fragments definition, all their parents and all model fragments which are imported to the copy buffer. The parents and imported model fragments are required as the definitions of the selected model fragments are incomplete without the existence of the parents and imported model fragments. Alternatively, the selected model fragments could be created by adding the model ingredients of the parents and imported model fragments explicitly in the model fragments. However, this is not desired, as the reuse aspect of qualitative models is lost. The model fragment selection (and the parents and imported model fragment definitions) can be pasted into another model by pressing CTRL+V. In this process, the selection in the model fragment hierarchy has no influence, since creating a model fragment underneath another model fragment, which is not its parent, alters the essence of the model fragment.

If a model fragment with the same name exists, the existing and the model fragment in the copy buffer are compared. The existing model fragment should be a superset of the model fragment in the copy buffer, if the model fragment is to be reused (meaning each element in the copy buffer model fragment should be in the existing model fragment). If this is not the case, a model fragment with the postfix '(copied)' is created.

6.7. Scenarios

Scenarios are less complex than model fragments, as they are not organised in a hierarchy, and contain less types of model ingredients. The implementation of the scenario list is comparable to the lists for some of the model ingredient definitions above. This makes it impossible to select multiple scenarios at the same time. The selection can be copied by pressing CTRL+C, which deletes the existing scenario definitions in the copy buffer, and copies the selected scenario definition to the copy buffer. Quantity definitions can be pasted to another model by pressing CTRL+V. If a scenario with the same name already exists, the paste procedure is treated in the same way as for model fragments.

7. Conclusions

This document describes the functionality added to the Garp3 workbench to enable collaborative model building.

The new Sketch environment can be used to capture intermediate model results. These intermediate results describe the domain at a higher abstraction level. By making these abstract ideas about the domain explicit, it is expected that it will be easier for multiple modellers to come to a consensus about issues in the domain. The formal representation makes the ideas modellers have about the domain more concrete.

The OWL export- and import-functionality allows models to be exported to an OWL format, and for models in that format to be imported again. The resulting OWL files can be stored in the online qualitative model repository, which is expected to become a central place to store and search for qualitative models. Making models available online makes modellers more visible to the community, allows modellers to look at previous work, and allows parts of previous work to be reused in new models (in combination with model merging functionality).

Multiple model support allows users to have multiple models open simultaneously. This is a requirement for model-merging support, as the content of multiple models has to be accessible to be able to transfer part of a model to another model. The model merging support makes it possible to copy/paste parts of models to other models. This functionality allows reuse of parts of existing modelling work.

Depending on user feedback, future work could include extending the OWL functionality to include the Sketch part of qualitative models. In that case the qualitative model repository will have to be adapted to allow search for models with specific Sketch ingredients. Furthermore, search for metadata will be added to the repository once more models are properly annotated.

8. References

1. B. Bredeweg, A. Bouwer, P. Salles and J. Liem. Towards a Structured Approach to Qualitative Reasoning. *20th International Workshop on Qualitative Reasoning (QR-06)*, C. Bailey-Kellogg and B. Kuipers (eds), pages 29-36, Hanover, New Hampshire, USA, 10-12 July 2006.
2. B. Bredeweg, A. Bouwer, J. Jellema, D. Bertels, F. Linnebank, and J. Liem. Garp3 - A new Workbench for Qualitative Reasoning and Modelling. *20th International Workshop on Qualitative Reasoning (QR-06)*, C. Bailey-Kellogg and B. Kuipers (eds), pages 21-28, Hanover, New Hampshire, USA, 10-12 July 2006.
3. Grigoris Antoniou and Frank van Harmelen. A Semantic Web Primer, The MIT Press, Cambridge, Massachusetts, April 2004.
4. Liem, J., and Bredeweg, B., 2006. *Document Type Definition (DTD) for QR Model Fragments*, Naturnet-Redime, STREP project co-funded by the European Commission within the Sixth Framework Programme (2002-2006), Project no. 004074, Project Deliverable Report D2.3.1.
5. 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.
6. J. Liem and B. Bredeweg. OWL and Qualitative Reasoning Models. In 29th annual German Conference on Artificial Intelligence (KI2006), 2006. *Lecture Notes on Artificial Intelligence.* Springer 2006. (To appear).
7. 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.
8. Bredeweg, B., Salles, P., Bouwer, A., and Liem, J. (2005) *Framework for conceptual QR description of case studies*, Technical report Naturnet-Redime, D6.1 (project number 004074), University of Amsterdam.
9. Bredeweg, B., Bouwer, A., and Liem, J. (2006) *Requirements analysis and re-design for collaborative workbench*, Technical milestone report Naturnet-Redime, M4.2 (project number 004074), University of Amsterdam.
10. Novak, J.D. and Gowin, D.B. (1984) Learning how to learn. Cambridge University Press, New York, New York.
11. Liem, J., and Bredeweg, B., 2006. *QR model-fragment library standard Redime*, Naturnet-Redime, STREP project co-funded by the European Commission within the Sixth Framework Programme (2002-2006), Project no. 004074, Project Deliverable Report D2.3.2.
12. Bredeweg, B., Salles, P., Bouwer, A., Liem, J. 2005. *Framework for conceptual QR description of case studies*, Naturnet-Redime, STREP project co-funded by the European Commission within the Sixth Framework Programme (2002-2006), Project no. 004074, Project Deliverable D6.1.