# Towards Interactive Tools for Constructing Articulate Simulations

**Vânia Bessa Machado and Bert Bredeweg**

Department of Social Science Informatics (SWI)
University of Amsterdam
Roetersstraat 15,
1018 WB Amsterdam,
The Netherlands
Email: {vania, bert}@swi.psy.uva.nl

**Abstract**

Having learners construct computer-based simulations is becoming increasingly important as an approach to induce learning. Qualitative simulations incorporate a rich vocabulary for articulating insights about systems and their behaviours, including notions such as structural constituents, identifying qualitatively distinct behaviours, and making explicit the causal dependencies that govern a system's behaviour. When building a qualitative model all these details have to be made explicit (they have to be represented in the model). As a result, building a qualitative model induces a deep understanding of the system and its behaviour. In our research we want to exploit this phenomenon, i.e. have learners learn by building qualitative simulations. However, building qualitative models is generally seen as a difficult task, and no easy-to-use tools exist to support a learner in performing such a task. In this paper, we present MOBUM, a domain independent model building environment aimed at supporting learners in building qualitative simulations. MOBUM is a fully implemented prototype in JAVA. The main goal of this prototype is to generate and clarify ideas with respect to how such a model building tool should be constructed.

## Introduction

In our research, we work from the hypothesis that the construction of models is an important aspect of learning. Within the context of simulations this means, 'having learners learn by building simulation models'. By constructing simulations learners can express their thoughts and exercise forms of thinking in order to reach a deeper understanding of systems and their behaviour (Forbus, 1996; Forbus et al., 1999; Soloway, et al., 1996).

Qualitative simulation models have characteristics that make them suited to support specific learning needs. For instance, when teaching to solve physics problems, teachers emphasise the need for learners to first understand the (problem) situation. Before trying to apply equations, learners should build a conceptual model (e.g. Mettes & Roossink, 1981) of the initial-state, the end-state, and the possible transition trajectories between the two. In fact, it is considered naïve (beginners-behaviour) if learners jump to 'applying formulae' without making a proper analysis of the problem situation (e.g. Elio & Sharf, 1990). Expert problem solvers excel because they spend a significant amount of time on making a conceptual model before using equations. Analysing (problem) situations is close to the idea of making an 'envisionment', that is, a mental simulation of what happens, or may happen (e.g. Kleer, 1984).

Qualitative models are also relevant in specific domains (often less formalised) where domain experts try to uncover the causal dependencies that govern a system's behaviour. After understanding the causal dependencies the experts may try to apply the mathematical formulae that are appropriate for the system. In fact, the causal model helps them to find the appropriate equations. Moreover, developing a conceptual model is often a goal in itself, that is: discovering the physical constituents of the system, identifying the relevant quantities, and understanding how these interact in determining the system's behaviour. Qualitative models are well suited to help domain experts in articulating and formalising their insights (e.g. Salles, 1997). If we think of how this kind of knowledge can be communicated to (new) trainees in the field, qualitative models are again crucial.

Qualitative models are sometimes referred to as 'articulate knowledge models', because they capture detailed insights with respect to how systems behave. Particularly, the notion of a causal interpretation of a system's behaviour is in this respect important (e.g. Forbus, 1988; Winkels & Bredeweg, 1998). To aid learners in acquiring these insights (i.e., by building simulation models) requires the development of computer software (i.e., tools) that support learners in constructing such articulate simulations. In this paper, we investigate the design and implementation of such a tool. The next section elaborates on the idea of articulate simulations and how those models are to be constructed. Section 3 and 4 describe the design of MOBUM, a fully implemented prototype that allows learners to build qualitative simulation models. Section 5 discusses related work.

## Building Articulate Simulations

Qualitative models can be seen as knowledge models of systems and their behaviour. For technical details of such models see e.g. Weld & Kleer (1990)[1]. Qualitative reasoning engines typically take as input a scenario and generate a graph of qualitative distinct behaviours for the situation described in that scenario. They use model fragments and transition rules to construct this graph (see figure 1). Scenarios are structural descriptions of the systems to be reasoned about. The idea behind model fragments is that each fragment represents a particular concept relevant to the domain that is being modelled, for instance: a population (ecology), a heat-flow (thermodynamics), or a pressure-area (meteorology). Transition rules are usually domain independent (e.g. a quantity taking on a new, higher, value from its quantity-space if it increases).

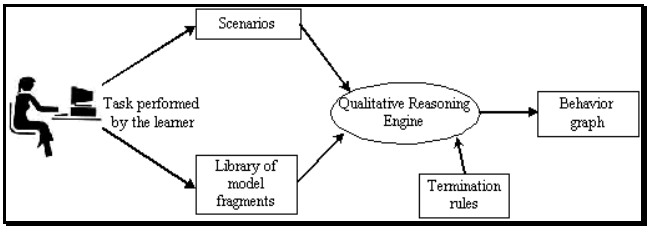


Figure 1: Building articulate simulation models

Knowledge engineers, or domain experts, are normally expected to construct libraries of model fragments. Once a simulation model is finished (i.e., all the necessary model fragments have been defined for a certain domain and the simulator can, on the basis of that, generate graphs of behaviour states for given scenarios) one can regard this result as an 'articulated knowledge model'. It captures the model builder's understanding of the domain, its systems and their behaviours. As such, the simulation model can then be used to communicate insights with other domain experts or it can be given to students to learn from. Notice that an important bottleneck with the construction of such articulate simulation models is the absence of easy-to-use, domain-independent, tools that support the model builder with this task.

## Constraints for the Design of Workspaces

Building a simulation model is a complex activity that involves many tasks, subtasks and interdependencies between tasks (e.g. Schut and Bredeweg, 1996). An important issue therefore concerns the decomposition of this overall activity into smaller parts that can be performed more or less independent from each other. These parts can thus be supported by separate, and possibly dedicated, interface constructs (referred to as workspaces). As a starting point we used the simulation model ontology on which GARP is based (Bredeweg, 1992) and remodelled it using an object oriented approach. A summary of this model is shown in figure 2. Without going into all the details of the figure, notice that the system model (that is the simulation model as a whole) consists of a hierarchy of model fragments, a hierarchy of entities, a scenario, quantity spaces and rules[2]. Both model fragments and scenario use 'descriptions', which implies that the latter have to exist (or have to be created) when creating a model fragment or a scenario. Entities and quantity spaces, on the other hand, do not use other model parts (but they are used as inputs when creating other model parts).

Description is a model construct used to group sets of model parts that are reused in other model parts. It turns out that these description parts are highly interrelated (partly shown in figure 2). For instance, a quantity always belongs to an entity, an attribute always exists between two instances, a proportionality always exists between two quantities, etc. Moreover, the specific descriptions that a learner may want to formulate always depend on the specific model fragment s/he is constructing. For instance, a liquid-flow process between two containers should only become active when those two containers exist, are filled with an amount of liquid, have unequal pressures and are connected by a pipe that facilitates the flow (see e.g. figure 5). Putting these insights together and making them available under the correct conditions is precisely what constructing a model fragment is all about. It is therefore necessary (i.e., logical) to have the interface facilities, for creating the descriptions, available in the context of creating a model fragment.

Constraints such as the ones discussed above have resulted in defining four builders (main workspaces) and sets of tools that are available within each workspace. Below, we enumerate the builders that exist in MOBUM.

- **Entity Builder**

In this workspace the learner models the (physical) objects that represent the domain. The hierarchical relationships between these objects will be modelled here as well.

[1] In our research we use GARP (Bredeweg, 1992), a simulator written in SWI-Prolog.

[2] In the current implementation of MOBUM a set of default rules (domain independent) is provided. Rules are therefore not further discussed in paper.

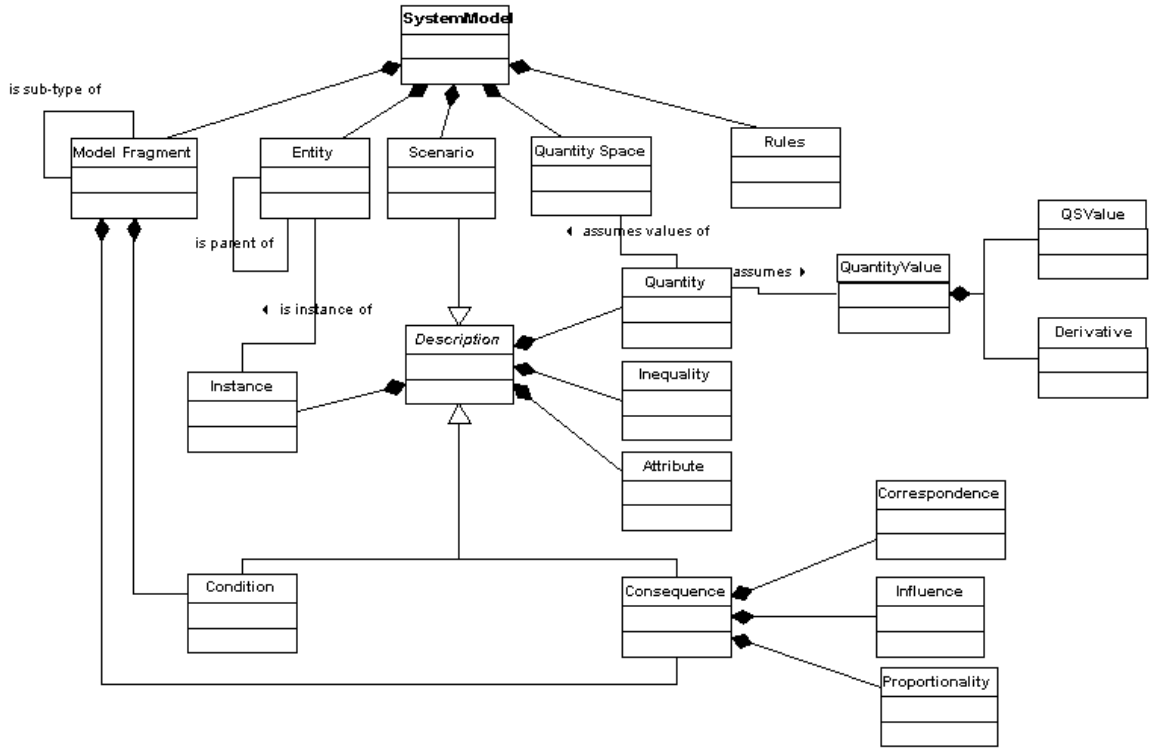


Figure 2: Simulation model ontology of the GARP qualitative reasoning engine

- **Quantity Space Builder**

In this workspace the learner creates an ordered set of quantity values that quantities may have. These values are a sequence of alternating points and intervals

- **Model Fragment Builder**

In this workspace the learner constructs the knowledge about the behaviour of entities. This includes the specification of features of instances, such as quantities, the values these have, and the dependencies that exist between the quantities.

- **Scenario Builder**

In this workspace the learner defines the situations that can be simulated. Notice that by definition this can only be a 'selection' of the model parts defined elsewhere in the model. For instance, there is no point in specifying an entity in a scenario that is not used in any model fragment.

Tools exist for creating, modifying and organising the model parts (mainly the descriptions in figure 2) within each builder[3]. For instance, table 1 enumerates the tools that exist within the model-fragment (MF) builder (see also figure 5, column on the right, read from top to bottom):

| | |
|---|---|
| Pointer tool | move icons on the screen with the workspace |
| Instance tool | add instances to the workspace |
| Modify tool | change a name |
| Delete tool | permanently remove something from the workspace |
| Attribute tool | make structural relations between instances, e.g. container contains liquid |
| Quantity tool | add a quantity to an entity |
| Influence tool | define a influence constraint between two quantities |
| Proportionality tool | define a proportionality constraint between two quantities |
| Correspondence tool | define a correspondence constraint between two quantities |
| Inequality tool | define a (in)equality constraint between two quantities |
| Reuse MF tool | add an already defined MF as a condition |
| Reuse instances tool | reuse parts of a conditional MF in order to further refine |

Table 1: Tools available in the model fragment builder

[3] Most tools are called 'makers' in the user interface of MOBUM. E.g. the quantity tool is referred to as the 'quantity maker' (see also figure 5).

Notice that, the Pointer, Delete and Modify tool are default tools, which are present in all builders. The tools that are used to compose a 'description', i.e. to place instances, attributes, quantities and inequalities are also present in the scenario builder.

To further constrain the design of MOBUM we used the notion of task analysis (e.g. Preece et al., 1994, see also figure 3). Mainly by detailing each of the subtasks within a workspace in terms of inputs-outputs, and thus their relative order, an additional set of requirements and constraints emerges.

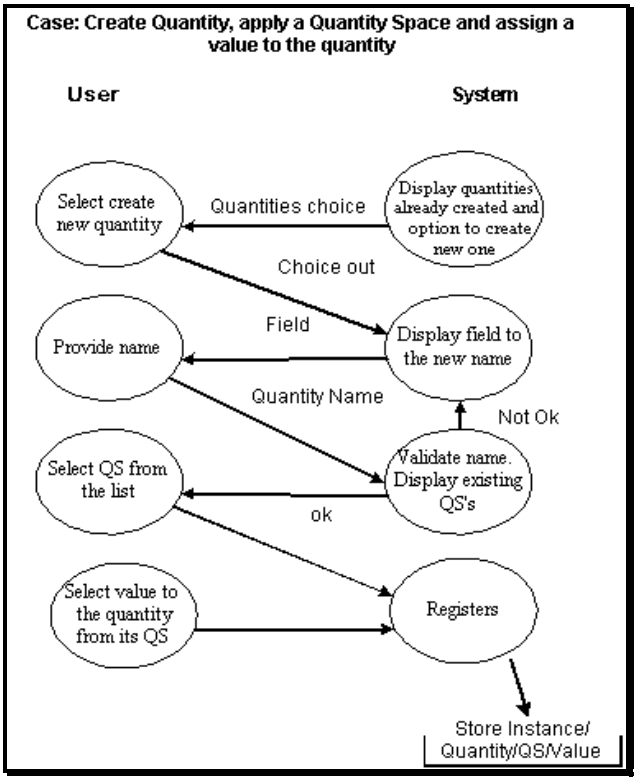


Figure 3: An example task analysis for making a new quantity.

Input-output dependencies determine whether a subtask can be performed and thus can be used to govern the availability of tools within a workspace. For instance, the quantity tool cannot be used unless at least one instance has been added to the workspace of the model fragment being built (and before that the learner must have created the entity as an element in the entity builder). Table 2 enumerates the basic requirements for using a tool within the model fragment builder.

When the basic requirements are fulfilled, and a tool can be used, the input-output dependencies within the subtask, supported by that tool, can be used to determine whether the learner has performed the task correctly or at least sufficiently (i.e., syntactically speaking). For instance, within the quantity tool the learner always has to select the instance to which the new quantity must be applied. The task is not sufficiently completed without that information and thus closing the task should be made impossible (of course, it can be cancelled). On the other hand, some information may not be crucial yet. For instance, the quantity space of a quantity may be added later. For each tool, the minimum required steps have been identified. They have been used in the design of the tools within MOBUM to support the learner in always performing the task to a sufficiently complete level.

| | |
|---|---|
| *Instance tool* | One entity must have been created (in entity builder) |
| *Attribute tool* | Two instances must exist (in workspace) |
| *Quantity tool* | One instance must exist (in workspace) |
| *Influence tool*<br>*Proportionality tool*<br>*Correspondence tool* | Two quantities must exist (as consequence in workspace) |
| *Inequality tool* | Two quantities must exist (in workspace) |
| *Reuse model fragment tool* | Other model fragment must be created (with MF builder) |
| *Reuse instances* | A reused model fragment must exist (in workspace) |

Table 2: Minimum requirements for using a tool in the model fragment builder.

## User Interface Design

The overall user interface design for the MOBUM environment starts with the notion of builders and tools. Tools are displayed on the right side of the screen and automatically change when the learner chooses to work with a different builder (i.e., workspace). Opening a builder can be done by clicking on the icons in the main toolbar, at the top of the overall interface. Builders can also be opened using the menu options. Multiple builders can be opened, but only one tool can be active. Although from a technical point of view multiple tools can be provided, the idea is that it is better to support learners in performing focussed model building steps. Allowing only one tool to be active forces learners to first finish the current task, or cancel it, before moving on to the next. Having multiple builders open is essential, because next to supporting a model building step, these builders also provide the learners with overviews of what has been built so far (see also figure 4).

Figure 4 shows a screenshot of a model building session in MOBUM. Two builders are open, the entity builder and the quantity space builder. The learner is working within the latter and uses the Point/Interval Maker. This is a tool that allows the user to add values to a quantity space. On the left side in figure 4 the model browser is shown (System model View). This browser provides an overview of the model building activities by showing the model parts that have been created so far. The browser can also be used for navigation and to open specific model parts and the corresponding builders (by double clicking on the name label).

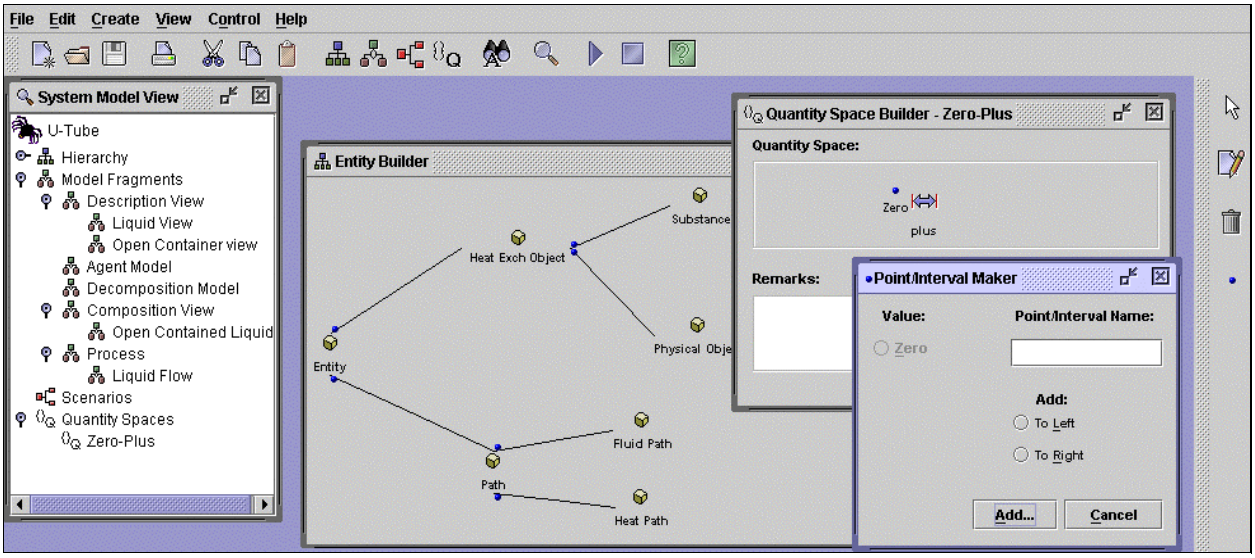


Figure 4: A MOBUM session, showing: browser, entity builder and quantity-space builder.

Next to the overall design, the internal design of the different workspaces has to be determined. Some choices are rather straightforward, such as using combo-boxes to present the user with a list to select from (e.g. instance selection in the Quantity Maker in figure 5). This is easier and prohibits typing errors. When the learner has to provide a new name, the words entered by the user are always checked against the already existing labels in order to prevent errors or undesired overlap. Less obvious design choices concern the icons and the spatial layout of some knowledge items on the screen. To start with the former, we have tried to find insightful icons to refer to knowledge items in a builder. For instance, an entity is visualised as a cube, a quantity as a gauge, etc. These choices are somewhat arbitrary. We do not yet know to what extent the MOBUM icon-language will be interpreted correct by the target users. However, most icons also have a label identifying them.

With respect to the spatial layout, most knowledge items (e.g. entities, instances, quantities, reused model fragments, points and intervals) are represented as nodes of a connected graph. They may be moved around freely by the learner allowing him/her to organise the model to his/her taste (using the pointer tool). An exception to this rule is formed by the quantities, which are organised in a tabular form, grouped together with the instance they have been assigned to. Included in this table are the quantity space, current value, and derivative for each of the quantities (see also figure 5).

Binary relationships between knowledge items are represented as lines connecting the two icons that visualise the knowledge items. The type of relationship is shown by an icon placed at the midpoint of the line (e.g., the > sign for an inequality, or an I+ sign for an influence, both shown in figure 5). A problem with this approach is that dependencies between quantities belonging to the same instance are somewhat difficult to represent, because they are lines 'from' and 'to' the same icon. Particularly in the case of 'many' lines this will become messy. Another approach would be to represent each of the quantities by a unique icon and then connect it, using lines, to the instance it belongs to. The problem with this approach is that it becomes more difficult to see what icons belong together. Also, in the case of 'many' quantities all the lines connecting all the icons make the overall picture pretty messy. More research is needed to find a good solution for this problem.

Another discussion concerns the 'conditions' and 'consequences' part of a model fragment. How to visualize this? In MOBUM we decided to have separate fields within the builder for this (see figure 5). Obviously, it is immediately clear whether an icon is in the conditions or in the consequences part. But there is also a problem and that is using the same knowledge item both as a condition and as a consequence. In the MOBUM approach this leads to a full copy of the instance (knowledge icon) into both the conditions and the consequences part, and then add quantities relevant to each part (in figure 5 pressure is added in the conditions part and amount plays a role in the consequences part). From the resulting visualization in model fragment builder it is not obvious that both icons refer to the same instance, the user really has to understand the underlying details. Another approach would be not to have separate fields, but to visualize the condition versus consequence role by means of a color. Again, more research is needed to resolve the issue.

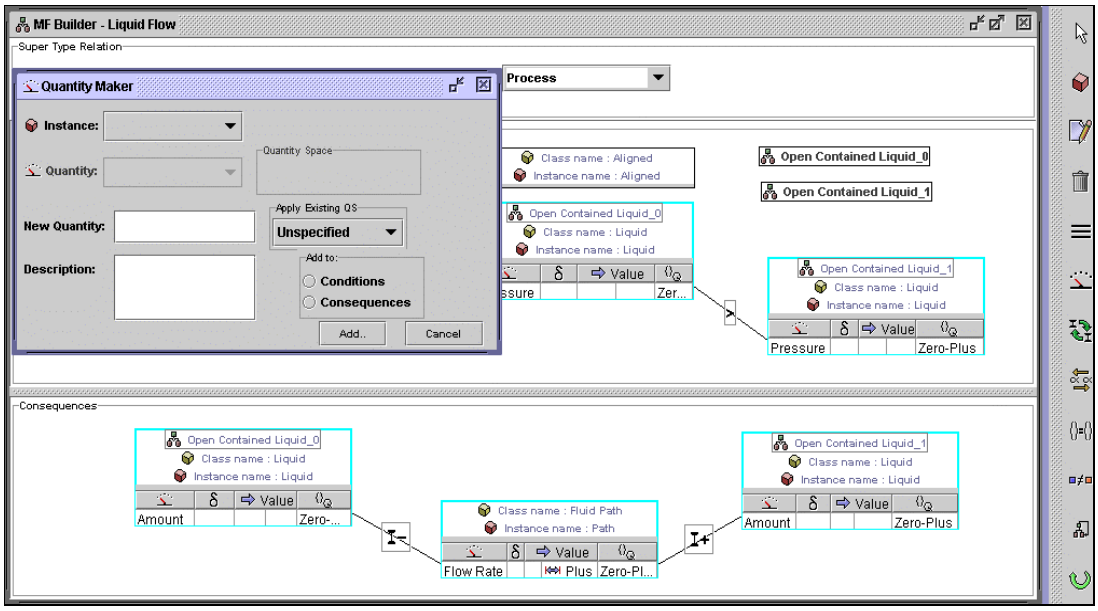


Figure 5: Model fragment builder: user starts adding a new quantity to the liquid-flow process.

## Related Work

MODEL-IT (e.g. Soloway, 1996) is a tool for creating simulations of dynamic systems, specifically ecosystems. In MODEL-IT, the learners are given objects (or s/he can make new ones) for the particular scenario at hand. The learner has to assign quantities (factors) to these objects and specify the behavioural relationships between the quantities of the different objects. By running the simulation, and maybe changing some objects/quantity relationships, the student can quickly explore the situation. MOBUM relates to the ideas underlying MODEL-IT, but there are also great differences. One difference is that in MOBUM the student has to work with full simulations, in which quantities can take on different values, different states will exist, model fragments may become inactive etc. The generation of a behaviour graph, while simulating, is not possible in MODEL-IT. Factors can only increase or decrease and there are no state changes. Also in MOBUM the idea is that learners construct model fragments that capture general insights, e.g. population growth (ecology), or heat-flows (physics), etc. In MODEL-IT the learner always makes a simulation for a specific scenario.

Another model building environment that should be mentioned is CYCLEPAD (Forbus et al., 1999), an 'articulate virtual laboratory' that supports engineering students in designing and analysing and thermodynamic cycles. The idea is that learners construct such systems by assembling components from a predefined library, i.e., as a set of icons on the screen, from which the learner has to select the ones relevant to the situation (scenario) for which a model has to be constructed. Selecting and connecting the parts is not always enough to get a simulation running adequately. Often learners have to provide additional detail, such as initial values, value-range limits and possibly modes of operation. Also remodelling the initially selected set of partial models may be necessary when the model does not produce the desired behaviour results. CYCLEPAD differs from MOBUM in that the former is domain specific, one cannot use it for ecology, for instance. Also, CYCLEPAD supports an *assembly* task (reusing predefined parts), which is different (and sometimes less difficult) from building a full model. In fact, CYCLEPAD does not simulate qualitative models, but rather quantitative models. The qualitative aspects are used to support the learner in assembling the system, i.e. for generating help.

Tools that support model building, but which are somewhat far from the ideas underlying MOBUM are tools such as MATLAB (Pratap, 1999) and STELLA (Grant, 1997). The main difference being the kind of models that learners might be able to build with such tools: quantitative simulation models. As such, these tools support the articulation of knowledge in a different format, which requires sophisticated foreknowledge of mathematics. However, some quantitative simulation-based tools notably THINKERTOOLS (White, 1993) and SIMQUEST (e.g. Hulst, 1996; Kuyper, 1998) provide features that in future research we may want to include in the design of MOBUM. These features concern the 'cognitive tools' that these tools have to support the learner in discovering the appropriate things, e.g., assignments, model progression, hypothesis scratchpad, etc.

## Discussion

In this paper, we have presented MOBUM, a model building environment for constructing qualitative

simulation models. MOBUM is a fully operational prototype that has been implemented in JAVA. The main purpose of the design, and implementation, of this first prototype was to generate and test ideas with respect to how such a model building tools should be constructed. In the current implementation of MOBUM there is no direct communication with the simulator. Instead, the learner has to save the model into files and then run the simulator (GARP) as a separate program. To be used in practice with actual learners this will not be sufficient. For that purpose, MOBUM and the simulator must be fully integrated, at least from a user's perspective.

MOBUM has not been evaluated with target users, instead knowledge engineers have commented on the usability of the current implementation (expert reviews). Three points for improvements have been pointed out. First, *intermediate modelling support*. Often when building a model, the persons building the model define intermediate models before they write down the final model. MOBUM does not support this process, it only supports the latter step. Improvements for MOBUM could focus on supporting and maintaining intermediate models for the learners. Second, *horizontal views*. When building a model in MOBUM it is difficult to see how all the model fragments that have been created will interact. There are no tools/builders/views that provide the learner with a global overview of certain model parts (except for the model browser). For instance, it may turn out to be helpful to provide the learner with a 'causal model viewer'. This would allow the learner to investigate if and how the causal dependencies, that have been defined in the different model fragments, are related (thus, without running the simulator first). Third, *model building support from a 'content' point of view*. MOBUM allows for building syntactically correct models, but the current prototype has no knowledge of the model construction process and the status of the model. In order to coach learners in building models beyond the syntactic level, MOBUM should be extended in this respect.

**Acknowledgments.** Vânia Bessa Machado wishes to express her gratitude to the Catholic University Dom Bosco, Campo Grande-MS, Brazil. She is currently on leave from this University to pursue her PhD at the University of Amsterdam.

## References

Bredeweg, B. (1992). Expertise in qualitative prediction of behaviour. Ph.D. thesis, University of Amsterdam, Amsterdam, The Netherlands.

Elio, R., & Sharf, P.B. (1990). Modeling Novice-to-Expert Shifts in Problem-Solving and Knowledge Organization. Cognitive Science, 14:579-639.

Forbus, K.D. (1984). Qualitative process theory. *Artificial Intelligence*, 24:85–168.

Forbus, K. (1996).Why computer modeling should become a popular hobby. D-Lib magazine.

Forbus, K. (1988). Qualitative physics: Past, present, and future. In Exploring Artificial Intelligence, pages 239-296, Morgan-Kaufmann.

Forbus, K.D. & Falkenhainer, B. (1992). Self-explanatory simulations: Scaling up to large models. In *Proceedings of AAAI-92*, pages 685-690.

Forbus, K.D., & Whalley, P.B., & Everett, J.O., Ureel L., & Brokowski, M., & Baher, J., Sven E. Kuehne, S.E. (1999). CyclePad: An articulate virtual laboratory for engineering thermodynamics. *Artificial Intelligence*,114: 297–347.

Grant, W.E., & Pedersen, E.K., & Marin, S.L. (1997). Ecology and Natural Resource Management: Systems Analysis and Simulation. John Wiley, New York.

Hulst, A. van der (1996). Cognitive Tools. Two exercise in non-directive support for exploratory learning. PhD thesis, University of Amsterdam, Amsterdam, The Netherlands.

Kleer, J. de & Brown, J.S. (1984) A Qualitative Physics Based on Confluences. *Artificial Intelligence,24*: 7-83.

Kuyper. M. (1998). Knowledge Engineering of Usability: Model-Mediated Interaction Design of Authoring Instructional Simulations. Ph.D. thesis. University of Amsterdam, The Netherlands.

Mettes, C.T.C.W., & and H. J. Roossink, H.J. (1981). Linking Factual and Procedural Knowledge in Solving Science Problems: {A} Case Study in a Thermodynamics Course. *Instructional Science*,10:333-361.

Pratap, P. (1999). Getting Started with MATLAB 5: A Quick Introduction for Scientists and Engineers. Oxford University Press, Oxford.

Salles, P.S.B.A. (1997). Qualitative models in ecology and their use in learning environments. Ph.D. thesis, University of Edinburgh, Edinburgh, Scotland, UK.

Schut, C. & Bredeweg, B. (1996). An overview of Approaches to Qualitative Model Construction. *The Knowledge Engineering Review, 11*(1): 1-25, 1996.

Soloway, E., Jackson, S. L., Klein, J., Quintana, C., Reed, J., Spitulnik, J., Stratford, S. J., Studer, S., Eng, J., & Scala, N. (1996) Learning Theory in Practice: Case Studies of Learner-Centered Design. *In ACM CHI '96 Human Factors in Computer Systems*, Vancouver.

Weld, D., & Kleer, J. de (Eds.) (1990). *Readings in qualitative reasoning about physical systems*. Palo Alto, CA: Morgan Kaufmann Publishers.

White, B. (1993). "ThinkerTools: Causal Models, Conceptual Change, and Science Education." *Cognition and Instruction*, 10(1): 1-100.

Winkels, R., & Bredeweg, B. (eds.) (1998). Qualitative Models in Interactive Learning Environments. *Interactive Learning Environment*, 5:1-134 (Special issue).