Model Based Systems and Qualitative Reasoning for Intelligent Tutoring Systems

International workshop at ITS 2002

June 2nd, 2002

San Sebastian, Spain

Preface

This workshop proceedings discusses the use Model Based Systems (MBS) and Qualitative Reasoning (QR) for Intelligent Tutoring Systems (ITS). The importance of MBS/QR for tutoring and training systems has been pointed out by many researchers in the area of Artificial Intelligence in Education (AIED). They agree on the necessity for rich, articulate and indexed simulations to facilitate a communicative interaction between learners and educational software. MBS/QR turns out to be important for the implementation of the major functions in intelligent training, help and teaching environments. An important goal of the workshop is to bring together the researchers interested in this area and discuss the future of MBS/QR for Educational purposes.

This is the fifth international workshop in a series of workshops dealing with research on how to use MBS/QR for the construction of educational systems. The first workshop was held during the international conference on AI in Education (AIED) in August 1995 in Washington DC (USA) and focused mainly on the use of qualitative reasoning techniques. Other workshops have been held at the international conference on Intelligent Tutoring Systems (ITS) in June 1996 in Montreal (Canada), at the European conference on AI in Education in October 1996 in Lisbon (Portugal) and at the European conference on Artificial Intelligence (ECAI) in August 1998 in Brighton (United Kingdom).

Bert Bredeweg

Amsterdam, May 5th, 2002

MONET (http://monet.aber.ac.uk/) is a European Network of Excellence on Model Based Systems and Qualitative Reasoning. MONET-2 is the successor of MONET-1 and active since January 2002. Within MONET-2 a taskforce focuses on the application of MBS/QR for educational purposes. This taskforce is partly responsible for organising the workshop at the ITS conference on this theme.

Programme Committee Members

Bert Bredeweg (chair)
University of Amsterdam (Netherlands)
http://www.swi.psy.uva.nl/usr/bert/home.html

Joost Breuker
University of Amsterdam (Netherlands)
http://lri.jur.uva.nl:80/~breuker/

Keith Brown
Heriot-Watt University (United Kingdom)
http://www.cee.hw.ac.uk/~keb/

Ken Forbus
Institute for the Learning Sciences (USA)
http://www.cs.nwu.edu/~forbus/

Riichiro Mizoguchi
Osaka University (Japan)
http://www.ei.sanken.osaka-u.ac.jp/english/menu.html

Chris Price
University of Wales (United Kingdom)
http://users.aber.ac.uk/cjp/

Paulo Salles
University of Brasilia (Brasil)
http://www.unb.br/

Julie-Ann Sime
Lancaster University (United Kingdom)
http://www.lancs.ac.uk/users/edres/personal/js/js.html

Elio Toppano
University of Udine (Italy)
http://www.dimi.uniud.it/

Contents

Investigating The Model Building Process with HOMER

Vania Bessa Machado and Bert Bredeweg
Department of social science Informatics (SWI)
University of Amsterdam
Roetersstraat 15, 1018 WB Amsterdam, The Netherlands
E-mail: vania, bert@swi.psy.uva.nl

Abstract

An experimental study is presented, which investigates the process of building qualitative simulation models using HOMER, a tool for building qualitative models of systems. HOMER consists of a number of dedicated editors aiming at decomposing the complexity of the model building process into more manageable subtasks. The aims of the present study are a) validating the original task analysis underlying the implemented system and the resulting task decomposition, b) identifying problems and/or misconceptions users encounter when building simulation models, c) assessing the tool's usability in supporting the model building process, and (d) defining possible improvements and faults of the present version.

Keywords: Qualitative Reasoning, Building Simulation Models.

1 Introduction

This paper presents the results of an analysis regarding the building of qualitative simulation models using HOMER [1] [2], a modelling tool for building qualitative models of systems, which can be simulated using GARP[3] . In the experimental study reported here, subjects used HOMER for building a model of a U-Tube system [4]. The subject's steps in solving the assignment together with the questions made during the experiment were recorded on video and subsequently transcribed for a fine-grained analysis. The protocols were analyzed and problems, misconceptions and modelling patterns were identified and clustered with regard to the conceptual aspects of the model building activities. Furthermore, problems encountered by participants due to the User Interface were registered for later improvement.

1.1 Building qualitative simulation models

Building qualitative simulation models is a complex process during which a multitude of aspects have to be managed by the model builder. At the most general level the problem of building a simulation model is to specify a set of model ingredients that can be used by a simulation engine to run a simulation in order to produce some required output (usually a particular graph of qualitatively distinct behaviors)[7]. This situation is sketched in figure 1.

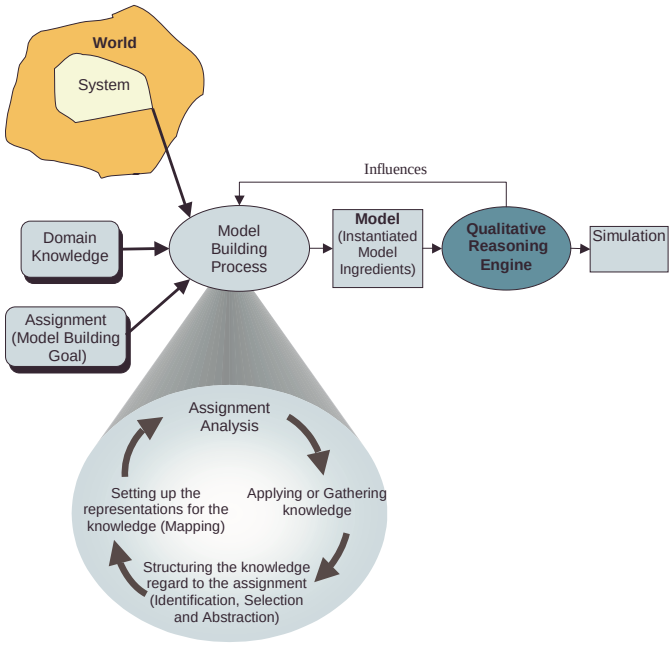


Figure 1: Model Building Process

The figure also shows that a building activity is particularly influenced by two aspects, the assignment and the reasoning engine. The former sets the requirements with respect to *what* must be captured by the model, where as the latter sets constraints with respect to *how* things can be represented in the model. Thus, when we look into the model building process with more detail, we can identify a cycle of activities with two main tasks distinguished. Usually, when initializing the model building process, a modeler analyzes the given assignment (or problem). Next, the modeler applies his (domain) knowledge in order to get the picture of the problem at hand. Certainly, if the modeler doesn't possess (domain) knowledge, s/he must gather knowledge about it before proceeding. Then, the phenomena from the real-world system must be identified, selected and abstracted into a set of system features that must be captured in the model. Finally these features must be mapped into model ingredients that obey the requirements of the simulation engine. Certainly, these activities are not independent.

Following the outline given above, specific aspects can be pointed out as being important for determining the result of a model building activity.

- Experience in model building in general. A model builder who is experienced in identification, selection and abstraction of real-world system phenomena, is in general better equipped to build models (regardless of the particular tool this person has to work with).

- Knowledge of model ingredients. The model building process will be easier, possibly better executed, when the model builder is more knowledgeable of the kind of ingredients that an engine requires.

- Complexity of the assignment (modelling goal). The complexity of an assignment depends on what and how many are the modelling ingredients the users have to work with and how users are supposed to organize/structure these model ingredients. Also, the existing ways of building the model are relevant: building by selecting, building by modifying and building from scratch. See [5] for a

description of these influential aspects.

- Representation of ingredients in a model building tool. Model building tools may differ with respect to how knowledge about system behavior must be represented. The way this is realized will influence the ease of model building.
- Knowledge of tool operation (traditional U.I. issues). The way a tool must be operated may be easy or more complex, for instance because it resembles operations used for other tools (or because it differs significantly from those known tools). Resemblance to known tools will probably make the use of the model building tool easier.

1.2 HOMER

HOMER is a tool for building qualitative simulation models. Technically, the HOMER environment (Figure 2) consists of a set of dedicated editors described as follows:

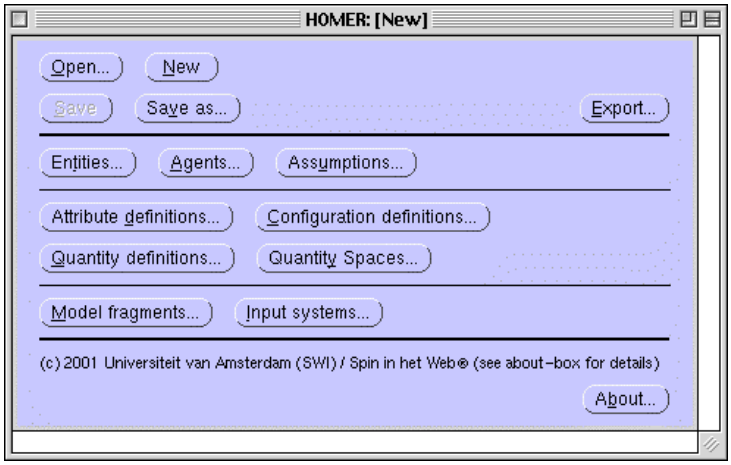


Figure 2: HOMER Main Window

- Entity Hierarchy: In this Editor the user models the (physical) objects that represent the domain. The hierarchical relationships between these objects will be modelled here as well, see Figure 3 for an illustration.
- Attribute definitions: In this Editor the user models the generic attributes that can belong to some physical object.
- Configuration definitions: In this Editor the user models the generic configuration that will be applied between two objects (Structural Relations).
- Quantity definitions: Here the user defines the generic quantities that may be applied to an object.
- Quantity Spaces: In this editor the user creates an ordered set of quantity values that quantities may have. These values are a sequence of alternating points and intervals.

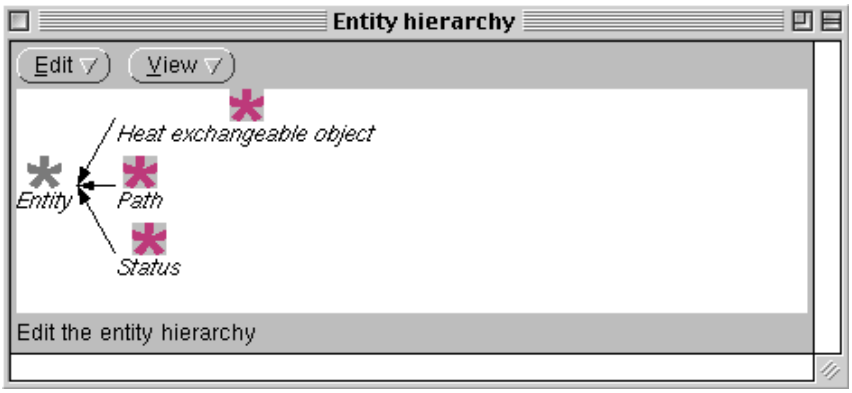


Figure 3: HOMER: Entity Hierarchy Editor

- Model Fragments: In this editor the user constructs the knowledge about the behavior of objects. This includes the specification of features of instances, such as quantities, the values these have, and the dependencies that exist between the quantities. In this editor, the user will be instantiating the generic model parts modelled in the previously mentioned editors (See Figure 4).

- Input systems: In this Editor the user defines the situations that can be simulated. Notice that by definition this can only be a 'selection'(instantiation) 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.

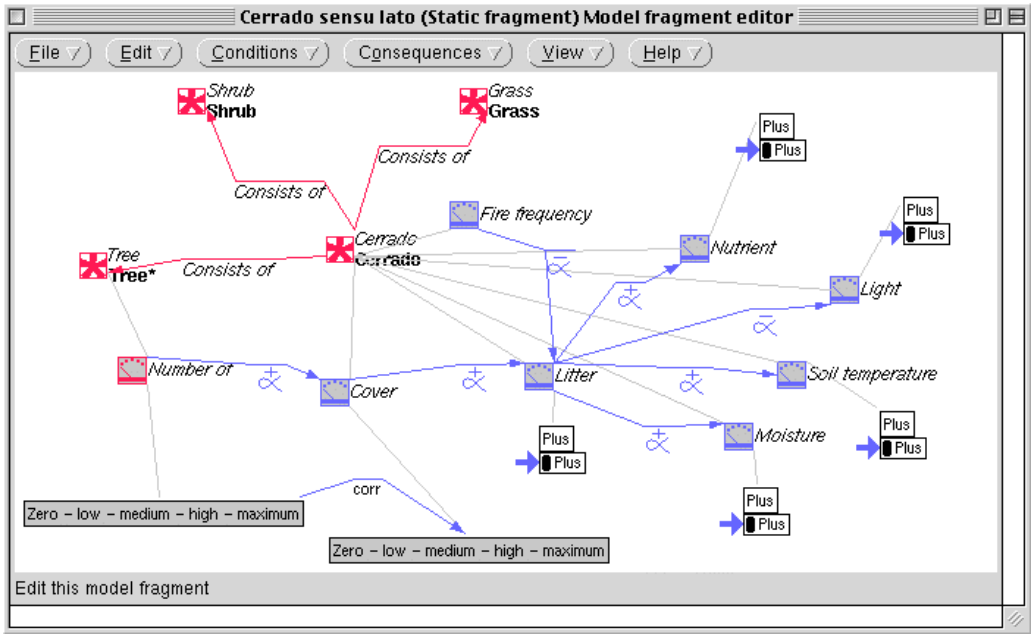


Figure 4: HOMER:Model Fragment Editor

Figure 5 displays the main sequence of usage of the editors. Notice that model fragments and input systems use model parts that come from the editors in the first line, which implies that these model

parts have to exist (or have to be created) when creating a model fragment or an input system. It turns out that these model parts are highly interrelated in the context of model fragments and scenarios. For instance, a quantity always belongs to an instance, an attribute always exists between two instances, a proportionality always exists between two quantities, etc.

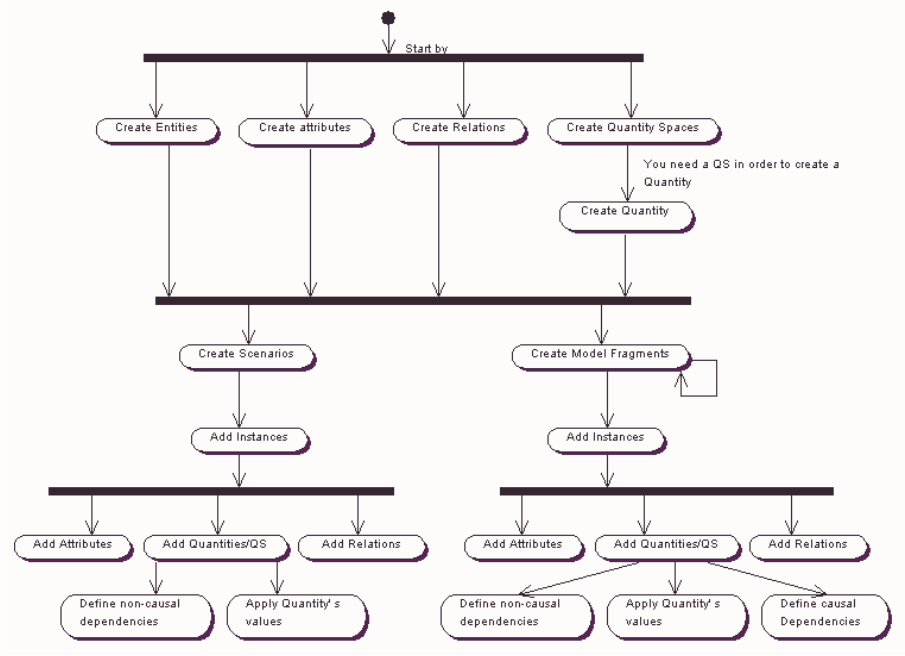


Figure 5: HOMER: Sequence of usage of the Editors

In this experimental study, our aims were a) validation of the original task analysis underlying HOMER and therefore, the resulting task decomposition, (b) discover what problems and misconceptions users encounter when building simulation models, (c) assessing the HOMER's usability in supporting the model building process, and (d) defining possible improvements and faults of the present version.

2 The Experiment

The experiment consisted of two parts. In the first part, the participants subjects should construct a simulation model of a U-tube system from scratch. The experimental setup consisted of handing out to the subjects a documentation containing the assignment, a short explanation of the qualitative modelling terms employed as well as a brief introduction to the HOMER environment. The second part consisted of giving specific tasks to the subjects as well as partial model constructs. All participants should start with the first part of the experiment. After thirty minutes if they didn't have advanced in the assignment of the first part the second part should be introduced.

Each session was recorded on video. The camera was pointed at the computer screen with the purpose of capturing the complete sequence of activities performed by the participants and therewith, gathering information about common behavior of a typical modeler. During the experiment, participants were allowed to ask questions, in fact we encouraged them to do so. Also, during the model building process participants were requested to think aloud. After completing the assignment, the subjects were asked to

give a summarizing reflection about the bottlenecks, flaws and possible improvements they encountered while working with the tool. This last step was recorded as well. Each session lasted approximately one hour. During the session, participants could use paper and pencil as well.

Participants

The participants were four people from our department (SWI). Two of them are researchers at our department and the other two are master students. All four participants had had contact with qualitative modelling.

3 Method

This experiment is about evaluating a model-building tool that supports the construction of simulation models. Figure 1 gives an overview of *all* the aspects involved in such a task, while figure 6 focusses just on those directly influenced by the support provided by the tool.

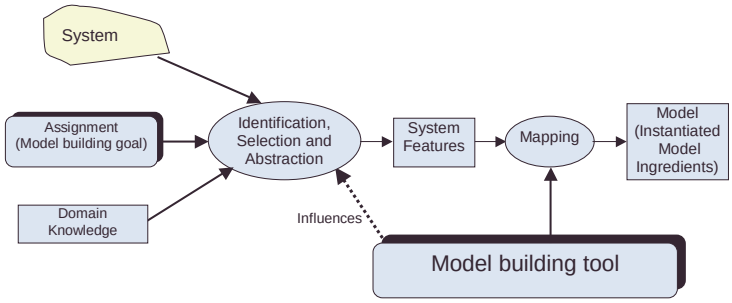


Figure 6: Model Building Tool

The aspects concerning the *Complexity of the assignment* and *Domain knowledge*, which also affect the model building process are not explored in the experiment. The reason for this is that the assignment is the same for all participants and each of the participants already possesses the required *domain knowledge* to solve the given assignment. These aspects therefore have little relevance in the outcome of the experiment. For similar reasons, the *assignment analysis* and *applying or gathering knowledge* steps of the model building process cycle(see figure 1) are also left out.

The doted arrow connecting the model-building tool to the process of identification, selection and abstraction indicates the overall influence that the tool may have on that process. Since the tool is based on a reasoning engine, which uses a specific modelling language, it is natural to expect that during the identification, selection and abstraction of the domain knowledge, the users will somehow be influenced by the possible ways of representing that knowledge.

The arrow connecting the tool to the mapping process represents the strong relation between the two. Following well-known user interface design patterns, the tool provides support by only making available correct modelling primitives, thus constraining the mapping of the system features in such a way that the output always results in a semantically correct model. There are several possible ways in which this support might be implemented. This experiment has thus also been used to evaluate the specific form of support proposed in HOMER. With this purpose in mind, we will firstly look at the task analysis and

its resulting products (see Figure 5 for an overview). Secondly, the focus of the evaluation turns to the major point of investigation within this experiment, which is discovering the model building problems encountered by the user while building a system model. Finally, based on important usability principles, we evaluate the User Interface. A discussion and extension of these issues is presented in the sections below.

3.1 Conceptual Aspects of Model Building Activities

3.1.1 Validation of the task analysis

The process of building simulation models involves a series of tasks and subtasks. The end goal of this process is the identification and representation of sufficient knowledge about a system in order to grasp its behavior. The objects found relevant for inclusion into the model and the way they are introduced may differ substantially from modeler to modeler. There are many correct ways of defining a model. Consequently, a multitude of possibly correct task flows may result, differing in the order in which the tasks are realized as well as in the hierarchical refinement into subtask.

In the *Task Analysis Validation* we validate the decomposition of the modelling process into seven major classes[1] (see Figure 5) which reflects the design decisions taken in the implementation of the evaluated tool. Mainly, we wish to know if the tasks and subtasks, and the order in which they occur, match the needs of the typical user.

Secondly, we want to sketch a profile of the typical modeler and understand the way in which the elaboration of a model is usually structured, if the available tasks were correctly interpreted by the subjects and if essential tasks were missing in the proposed task flow.

3.1.2 Model Building Conceptions

Model Building Conceptions focus on the identification of the participants' problems in performing the following (sub)tasks:

1. **Scoping the Model:** Identification of the knowledge relevant for the system at hand (In this experiment the *U-Tube System*). E.g., What are the relevant quantities in the system?

2. **Structuring the Model:** Organization of the model (the relevant knowledge) into a working simulation model which can be used to predict the system's behavior. This step is quite critical. It reflects the participant's view of how to represent the desired system. An example of the questions that might have to be answered at this stage is: How many model fragments are needed to represent the U-Tube System?

3. **Model Building Concepts:** Understanding the concepts that constitute the model of a system. E.g., What is the difference between *attributes* and *quantities*?

3.1.3 Representing The Model

Representing the model refers to representing the system's phenomena in terms of the available ontology for building simulation models. It may happen, for instance, that the modeler knows what he wants to say

[1] Entity, Attribute, Configuration, Quantity, Quantity Space, Model Fragment and Input System

but he doesn't know how to say it using the available ontology. Representing the model (Mapping) plays an important role into the cycle of model building activities. If the user assembled the system features but still doesn't know how to represent the features in the ontology for building simulation models they will not proceed in the cycle. Therefore, a model building tool should help the users in this aspect, such as, by preventing them to perform erroneous mappings.

3.2 User Interface - Overall Usability

In this experiment the subjects were required to go through the process of building simulation models. In order to do so, they had to learn to manage a new tool, making the task more complicated. A proper tool should facilitate the users' comprehension and interaction regarding the solution of the problem. Usability thus, plays a crucial role in attaining one of the major goals in a successful interactive system. The *Heuristic Evaluation* method is used to describe the problems arising during the subjects interaction with the interface. By considering Nielsens ten usability heuristics[6] we interpreted the user's actions in order to infer in how far these actions are related to usability issues in the interface's design.

4 Results

The experiment consisted of two parts. However, it turned out that all the subjects performed satisfactorily on the first part of the experiment, so that the realization of second part became unnecessary. Since the official time to complete the experiment had already past, one participant didn't complete the task of creating an input system (scenario). In the following, the main results of the experiment are summarized and categorized into three main classes: Conceptual Aspects of Model Building Activities (*Task analysis, Model Building Conceptions)*, Representation of the Model and User Interface Evaluation.

4.1 Task Analysis

Sequence of activities

Despite the fact that the existing activities in the highest level of the tool are concurrent (See figure 5), the participants started by creating the hierarchy of entities. Just one participant, Raichu, didn't create a hierarchy of entities. However, he gave meaningful names to the entities when including them in his model fragments. By doing so, we may conclude that he was aware about the existing entities in the U-Tube system but he wasn't aware that the definition of these entities should be done in a hierarchical manner and in a dedicated editor.

In summary, examining the data, we can conclude that, essentially, the mainstream sequence of activities was: Creation of a hierarchy of entities in the first place, followed by the creation of the model fragments and finally the input system. Together, these three concepts constitute the main building blocks in GARP and were also the ones, the subject focused on constantly. Other concepts, such as configurations, quantities and dependencies played a more secondary role as they were more loosely coupled and appeared only in the context of model fragments and scenarios. Although, in the context of Model Fragments and Scenarios, the sequence of performed (sub) tasks was determined mainly by the UI (For instance, the user couldn't add a quantity to the model fragment if an entity, which the quantity

belonged to, was not selected.), still there was a mainstream sequence of activities 1)Add Entities, 2) Add Configurations, 3) Add Quantities and 4) Add Dependencies.

A significant deviation of this sequence happened when some *repair* had to be made. Notice that this sequence of activities partially matches the ideal sequence of activities implicitly suggested by the tool. Looking at Fig.5, we can see that, at the highest level we have the (sub)tasks of creating entities, attributes, relations and quantities which matches the sequence in which the participants' completed the various tasks. However it is clear that the creation of Quantity Spaces was not seen as an independent task by any participant as it is suggested by the tool.

Understanding the activity that should be performed

The participant didn't have problems in understanding the activities.

Missing tasks

Although, these were not the main problem in completing the task, some missing tasks were noticed or pointed out.

- Two of the subjects made draws before specifying their models. This can be indicative that a support for this task should be part of the tool
- The subjects pointed out that they missed a complete overview of the model constructed so far. It was mentioned that if they were modelling a complex system it would be harder to understand their models (For instance, because of the crossing lines, hierarchy of Model Fragments, etc)
- An interesting aspect noticed was that the subjects were often constructing a mental model of the behavior of their system, particularly concerning how quantities are causally related. This suggests that the tool should support the causal model view (editor) constructed so far.

4.2 Model Building Conceptions

Scoping the model

None of the participants had difficulties in specifying the U-Tube system's entities. Remarkably, our data analysis shows that all participants experienced difficulties in defining which quantities were relevant for the model. Also, participants effort were in defining the relevant values for a quantity space, defining relevant model fragments and defining when the model building task was finished (all the relevant knowledge was specified). E.g.,

- E.g.:"What quantities to define?
  Maybe we need pressure!"
  "I don't have any quantity. At least we need (quantity) *pressure*. I don't know".
  "Do I need to model *pressure* and *flow*? I need *flow*. Maybe I need *pressure*"
- "I am not sure if I need the value *max*..."
- "Do I need another Model Fragment to define flow?"

- "Am I ready now?"
  "The main problem was in determining when you are finished. You need to be aware of what you have already done and what you still need to do".

Structuring the model

The participants were not sure about *structuring* their models. Sometimes they were unclear about where to add a knowledge item in the model. For instance, a participant was confused about when to specify a quantity's value in a model fragment. Similarly, participants were in doubt about which quantities should be added in a specific model fragment. Still regard to the quantities, participants were in doubt about which quantity space should be given to a quantity and, also defining which entity holds a quantity. E.g.,

- "I didn't define values in the Model Fragment. Do I need to do it for the model fragment be applicable?"
- "I don't know what QS to give to height and I want to say that one height is bigger than another"
- "(Looking at the mf) I don't know if I need the quantity *flow* in this model fragment."
- "I wonder if I need *(quantity) Level* as properties of the container or the liquid entity"

Model Building Concepts

Sometimes the subjects lacked an understanding of the important model building *concepts*. A participant was confused about the meaning of the concepts *attribute* and *quantities*. Another time, a participant wasn't aware of the difference between the conditions and consequences in a model fragment. A participant didn't know the relation between instances in a scenario with the ones in the model fragments. Still, a participant was confused about the difference between two quantities with the same quantity space values. E.g.,

- "Container has *height...*"
  then, the participant selected *Attribute definition*. Therefore, when he should specify the attribute' values the following happened: "Value (?) - I don't know anything about value. I just want to say that the height of one container is bigger than the other." [2]
- Difference between two quantities with the same QS Values
  E.g.:"Are quantities with the same values-name equals?"
- "It was not clear to me that conditions in the model fragment include entities"
- "should the names of entities in the input system match with the names given in the MF?"

4.3 Representing the Model

The results also show that the subjects had problems in specifying the knowledge using the available ontology. During the experiment, a subject tried to add a configuration (relation) between two quantities. This shows that there was a confusion of the meaning of configuration once that this can be done just

[2] At this point the experiment supervisor intervened and explained that what he meant was a quantity

between entities. Also, in some cases, subjects did not understand the conceptual difference between an inequality and a proportionality. Some examples are given below.

- "How do I say that there is a Flow due to the difference in the amount of water?"
- "I want to represent that *PressureDifference* is the difference of the two (quantities) *levels*."
- "Now I expect proportionality. I mean the pressure difference is proportional to the level difference"

4.4 User Interface

This section summarizes the evaluation by emphasizing the more significant usability problems. The evaluation and judgement of each concern or problem is done in compliance with Nielsens ten usability heuristics.

4.4.1 Visibility of the system's status

The "New" button in the main window mislead two of the subjects. As the system doesn't show that the user is working on a new model when it is initialized, the subjects wondered what the status of the system was.

Inside the Quantity Builder, when intending to create a new quantity, while all subjects typed in the quantity's name into the name field, they failed to realize that the quantity would only be created by pressing on the "Add" button. It sometimes happened, that the user thought s/he was creating another quantity, when s/he actually was editing the previous one.

4.4.2 Match between system and the real world

In the main window the menu selections matched the terms commonly used by the subjects. The "Configuration definitions" option was one exception. This option enables users to specify relations between entities but it was not clear to the subjects.

Usually, when specifying a correspondence, it should suffice if one says that Quantity A corresponds to Quantity B. In HOMER, matters are somewhat complicated by the fact that a correspondence has to be specified by the selection of the associated Quantities' Quantity Spaces. This feature also was confused by the participants.

4.4.3 User Control and Freedom

The user must use the system as it is, there is no customization available.

4.4.4 Error Prevention

The *Error Message* in the creation of a Quantity without an associated QS confused the subjects. Especially after having invoked the QS Editor from within Quantity Editor in order to create the quantity's quantity space. They reckoned it should have been automatically associated to the Quantity that was being created. Frequently, this misunderstanding lead them not to finish the task or to loose the quantity entirely.

When adding an object to a model fragment or scenario the last added element is always selected by default, which makes the subjects to add some elements to the wrong font.

4.4.5 Consistency and Standards

Looking at the Editors from the highest level, only in the Quantity Editor the user cannot create an object *per se*. In order to create a quantity it must be associated with a Quantity Space. This caused a confusion among several of the subjects.

In the Entity Editor one of the subjects expected a "New" button in order to create a new Entity. There is none. Instead there is the "Add Child" button. At the same level in the others editors, however, the subjects noticed that there is a "New" button. According to them this was not consistent.

4.4.6 Recognition rather than recall

When adding objects (attributes, configurations and quantities) to the Model Fragments or to the Input System, the user has the option of editing these objects inside their original editors. However, with the resulting proliferation of windows, the users would get lost and not know exactly at which step of the creation of their model they were. Also, the distinction between the Editor and the Creator was not always apparent.

4.4.7 Flexibility and efficiency of use

None of the users realized how to proceed in order to set up a value for a quantity.

4.4.8 Aesthetic and minimalist design

In a Model Fragment and Input System, the option of hiding and showing information wasn't explored by the subjects. Therefore, when adding a quantity all information concerning it (Quantity Space and Derivative) was also added to the window even if it was not relevant. Moreover, two of the subjects complained about too much information on the screen.

4.4.9 Help users recognize, diagnose, and recover from errors

Most of the errors were solved by repeated trial and error attempts. The system is rather deficient in providing help at these times. On several occasions the experiment's supervisor had to intervene in order for the subjects to continue. A subject selected three objects to add a dependency to. There isn't any help in regard to this error.

4.4.10 Help and documentation

A printed version containing Help about the system's functionality and model building terms was available to the subjects. However, none of them did consult it.

5 Discussion And Future Directions

Our major goal in performing the experiment presented in this paper was in investigating the main problems users encountered when building simulation models. Additionally, we wanted to validate the task analysis underlying the implemented system and the resulting task decomposition.

HOMER provides all necessary modelling primitives as well as support to assist the user in the model building process. Before HOMER, the only way to build simulation models (in the context of GARP simulation engine) was by using text files. In this way, a lot of expertise is demanded from the builder not only in formalizing his model correctly but also in taking care of syntactic details such as comas, parenthesis etc. HOMER allows graphical representation of concepts and their relationships. In addition, it provides the means of representing knowledge in a simple and intuitive visual form. The results shows that the visualization tools help users focus on performing a specific task.

Based on the patterns in participants data noted above, two features will be the focus of our future research. First, the design of a support module that can guide the users through of the model building process. The support module is intended to reduce the cognitive load as well as to give to the user more confidence in the process of model construction. To be used in educational settings, the support module should incorporates support in terms of model content and also knowledge about the system status. It means, being able to keep track of the users actions, reasoning about them and providing feedback to the users. Second, the design of a flexible model building tool that can support and maintain intermediate models for the user and, also providing the user with a global overview of certain model parts. For instance, showing how all the model fragments that have been created will interact and also, it was apparent in the results that is helpful to provide the user with a 'causal model viewer'. This would allow the user 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). Additionally, all the improvements related to the user interface usability and functionality should get attention in a new design.

References

[1] Homer 2.0 gebruikers documentatie. Technical report, SWI/UvA, 2001.

[2] Homer 2.0 technische documentatie. Technical report, SWI/UvA, 2001.

[3] B. Bredeweg. *Expertise in qualitative prediction of behaviour*. PhD thesis, University of Amsterdam, Amsterdam, The Netherlands, March 1992.

[4] K. D. Forbus. Qualitative process theory. *Artificial Intelligence*, 24:85–168, 1984.

[5] V. Bessa Machado. Towards understanding model building complexity. Technical report, SWI/UvA, 2001.

[6] J. Nielsen. *Usability Engineering*. Academic Press, San Diego, 1998.

[7] C. Schut and B. Bredeweg. Supporting qualitative model specification. In *Proc. of 2nd International Conference on Intelligent Systems Engineering*, Technical University of Hamburg, Hamburg, Germany, September 1994.

Aggregation of Qualitative Simulations for Explanation

Anders Bouwer & Bert Bredeweg

Department of Social Science Informatics, University of Amsterdam
Roetersstraat 15, 1018 WB, Amsterdam, The Netherlands
E-mail: {anders, bert}@swi.psy.uva.nl

Qualitative simulations can be seen as knowledge models that capture insights about system behaviour that should be acquired by learners. A problem that learners encounter when interacting with qualitative simulations is the overwhelming amount of knowledge detail represented in such models. As a result, the discovery space grows too large, which hampers the knowledge construction process of the learner. In this paper we present an approach to restructure the output of a qualitative reasoning engine in order to make it better suited for use in interactive learning environments. The approach combines techniques for simplifying state-graphs with techniques for aggregating causal models within states. The result is an approach that automatically highlights the main behavioural facts in terms of simulation events and simplified causal accounts, while leaving the option for the learner to explore the aggregated constructs in more detail.

Keywords: Qualitative Simulation, Model-Based Reasoning, Automated Abstraction, Explanation Generation, Intelligent Learning Environments

1. Introduction

This paper addresses the problem of making qualitative simulations of complex systems easier to understand for learners. The simulations generated by qualitative reasoning engines are often difficult to understand, because these models capture a lot of detail about the structure and behaviour of a system.

Previously proposed solutions to this problem can be grouped into two classes. One group tries to generate less complex simulations to begin with, while the other group tries to summarize the results of complex simulations afterwards. In the case of the former, the idea is to use *information needs* (such as user questions) as guidance. For instance, given a particular question about the behaviour of some system, a parsimonious simulation can be generated that does not necessarily account for all possible behaviours of that system, but that is sufficiently detailed to address that particular question (e.g. Falkenhainer & Forbus, 1991; Rickel & Porter, 1997). Mallory et al. (1996) present ideas within the second group of approaches. By analysing the *behaviour paths* in a state-graph for certain features, multiple states can be grouped into a 'single' state, simplifying the graph as a whole. De Koning et al. (2000), also within the second group, take a rather different approach when they aggregate the causal model within a state. Although this approach significantly reduces the complexity of the causal model, the link with important state-graph features (as discussed by Mallory) is missing, because the aggregation is always applied within a single state of behaviour.

We view a qualitative simulation as a knowledge model that captures certain insights that a learner should acquire. This model, made by a teacher or with the help of a teacher, must therefore be treated as a given, it cannot be reduced beforehand. This implies that we need an approach from the second class, namely one that takes the output of a reasoning engine and makes it easier to understand for a learner. Specifically, we combine the ideas from Mallory et al. (1998) and De Koning et al. (2000), and use them not only to construct a simplified state-graph, but also to create a simplified account of that graph in terms of the underlying causal relationships. In addition, we provide the learner with the possibility to open up the aggregated constructs, so that the learner can also explore the simulation results in more detail. Our approach can be regarded as a hierarchically structured simulation model that simplifies the discovery process for learners by highlighting the important behavioural facts.

The content of this paper is as follows. Section 2 introduces a taxonomy of events, describing how the behaviour of a simulated system can be analyzed hierarchically in terms of events. Section 3 explains how the notion of events on different levels of aggregation can be used to select interesting information while abstracting from the rest. Section 4 discusses the differences with respect to previous work, as well as directions for further research.

	State Graph Events	Value Events	Inequality Events	Structure Events	Model Fragment Events	Causal Events
Global Simulation Level	Start and end states Reuniting of paths	Different path behaviours Common behaviour Global max/minimum	Different path behaviours Common behaviour	Different path behaviours Common behaviour	Different path behaviours Common behaviour	Input can lead to any end state Different path Behaviours Common behaviour
Path Level	Sequence of transitions Recognition of branches	Sequence of events below Repetition of events below Path max/minimum	Sequence of events below Repetition of events below	Sequence of events below Repetition of events below	Sequence of events below Repetition of events below	Begin may lead to end state Sequence of events below Repetition of events below
Path Segment Level	Sequence of transitions	Sequence of events below Repetition of events below Segment max/minimum	Sequence of events below Repetition of events below	Sequence of events below Repetition of events below	Sequence of events below Repetition of events below	Begin leads to end state Sequence of events below Repetition of events below
Local Level	Outgoing branch Incoming branch	Reach value and stay Move from value Cross value Reach extreme value	Become equal Become greater Become smaller	Entity (dis)appears Entity changes Attribute (relation) (dis)appears Attribute (relation) changes	Situation becomes (in)active Situation changes Process becomes (in)active Process changes	Qx has pos./neg./no effect on Qy dQx has pos./neg./no effect on Qy
State & Transition Level	Momentary states Interval states Momentary transitions Interval transitions	Value Derivative Value transition	Equality Inequality Inequality transition	Entity exists Attribute exists Attribute relation exists	Process is (in)active Description view is (in)active (De)composition view is (in)active Qualitative state is (in)active	Pos/neg. influence of Qx on Qy Pos/neg. proportionality from Qx to Qy

Figure 1. Event types at different levels of aggregation.

2. A taxonomy of simulation events

Our goal is to make detailed descriptions of system behaviour easier to understand for learners, by pointing out patterns, and abstracting information using such patterns. As the basis of our method, we decompose behaviour into *events*, which can be causally and temporally related. We distinguish different kinds of events, both in terms of *categories* (the type of information) and *aggregation level* (the degree of abstraction). To this end, we have devised a taxonomy of events, as shown in figure 1. The contents of the matrix figure will be discussed in detail in the next two subsections.

2.1. Levels of aggregation

The rows of figure 1 denote the different levels of aggregation, which will be discussed from bottom to top, because this is the order in which they are derived.

State and transition level: this level contains state descriptions and transition specifications, the representations used by the simulation program, in our case GARP (Bredeweg, 1992). Each state specifies the structural elements and relations which hold at that moment or time interval, the quantities along with their values and derivatives, the mathematical and causal relationships between quantities, and the active model fragments (representing situations, or processes). Transitions specify essentially which quantity values or inequality relationships change (in a qualitative sense), but domain-specific rules may be added to introduce structural changes as well (e.g., the lid of a container may open when the pressure exceeds some threshold). Because the information at this level is the output from the simulator, it functions as the bottom level for the higher levels of aggregation.[1]

Local level: this level comprises two (or three) states and the transition(s) in between. It largely corresponds to Mallory's notion of *trajectory* (1996), although he focuses mostly on value and derivative events, while we include other types of events as well. Like the transitions on the previous level, this level deals with the difference between adjacent states, but it does this in a more integrated way. A transition specifies the basic changes from one state to the next, in terms of quantity values and inequality relationships, but it does not fully specify the successor state. Hence, not all the differences between adjacent states (like the situation description changing, or processes becoming active) are included in a transition. On the local level, also these changes are explicitly represented.

Path segment level: this level aggregates successive events until multiple possibilities arise, i.e., a branch occurs in the state-transition graph. This level contains the same types of events as the path level (the next level), but it is interesting as an intermediate level for explanation purposes. When a simulation contains a path segment (i.e., a state sequence without branches) from s_m (possibly via s_{m+1}, s_{m+2}, …) to s_n, we can say that the situation at s_m has led to the situation at s_n. When we consider a longer path which includes a branching point, e.g., the path s_m to s_{n+1a}, while there is another transition from s_n to s_{n+1b}, we can no longer say that s_m has led to s_{n+1a}, because it could have led alternatively to s_{n+1b} (see figure 2).

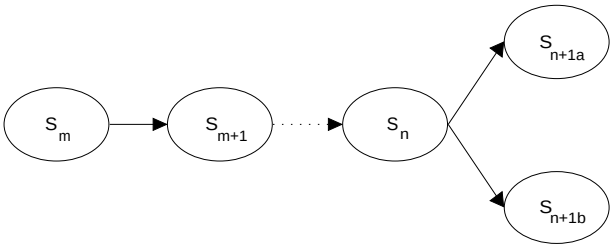


Figure 2. Path segments vs. paths. The path from S_m to S_n is also a path segment, but the path from S_m to S_{n+1a} (or S_{n+1b}) is not, because it includes a branching point.

[1] While calculating the simulation results, the program uses even lower-level internal representations, but these are not interesting for our purposes.

Path level: this is a generalization from the path segment level, allowing branching points to be included. However, like the path segment, a path is still a strict sequence of states and transitions, so for every branching point, only one of its successors is included. About a path, one can say that the begin situation *may* lead to the end situation. Also, for a path, one can talk about repetitive, or cyclic behaviour, if present. When the begin and end of a path are also global begin and end states in the simulation, such a path represents a possible behaviour of the system simulated.

Global simulation level: this level includes all alternative paths, if there are any. If the simulation results in only one path, there is only one possible behaviour of the system modeled, given the input to the simulator. In many cases, however, the input or the model does not fully specify the behaviour in advance, and multiple possibilities will arise, creating branches in the state-transition graph. At the global simulation level, we can look at the differences between such alternative paths. In some cases, alternative paths only differ with respect to the order in which events occur, but in other cases, alternative paths may contain very different events altogether.

2.2. Event type categories

With these levels of aggregation in mind, we now focus on the different event type categories that we distinguish in the different columns of figure 1.

State graph events: on the *state and transition level*, states and transitions can be momentary or take some interval of time. On the *local level*, it's also possible to recognize branching points when there are multiple transitions from, or to the same state. On the *path segment level*, there are no branches. On the *path level*, there can be branching points, and also cyclic behaviour is recognized. On the global level, paths branching out and reuniting again can be recognized, as well as global start and end states.

Value events: at the *state and transition* level, every state specifies the value and derivative of every quantity in that state. On the local level, Mallory et al. (1996) has introduced some event types in this category. The information of two states is combined to form events, such as *reach value and stay*, or *move from value*. For some events, it's necessary to consider three successive states, because they form a more natural whole than any combination of two, e.g., a *maximum* requires a state in which a quantity value is increasing, a state in which it is steady (this may be a momentary state, or a state lasting for an interval of time), and a state in which it is decreasing again (Mallory et al., 1996). At the *path (segment) level*, these events can be further aggregated by chunking continuous or repetitive developments, abstracting where necessary from local maxima and minima to path (segment) level extremes. At the *global simulation level*, the differences and commonalities between different behaviours can be determined, as well as global extremes.

Inequality events: in the individual states, (in)equalities are specified between pairs of quantities, if applicable. In the state transitions, changes in these (in)equalities are specified. On the *local level*, these are essentially preserved, with an exception for the case of a continuous change from Qx < Qy via Qx = Qy to Qx > Qy (or vice versa): this is chunked to a change from Qx < Qy to Qx > Qy. This kind of chunking may also occur on the *path (segment) level*, if the changes are spread out over more than three consecutive states.

Structure events: every state specifies the structural constellation of the system modeled, in terms of entities and relationships. Whenever an entity or relationship (dis)appears, or changes, this constitutes an event on the *local level*. Since our qualitative simulation engine is geared towards representation of change in terms of varying quantities rather than spatial information, not much further aggregation is possible in this category.

Model fragment events: model fragments specify situations and processes, although the *state and transition level* contains a more fine-grained typology. When its conditions are met (specifying structural, value or (in)equality constraints), a model fragment becomes active in a particular state, potentially introducing more information. A *process* model fragment typically introduces a *flow* quantity influencing other quantities which are often involved in the triggering condition (e.g., T1 > T2 introduces a heat flow). On the *local level*, processes can become active or inactive, and situations may change. Since model fragments are organized in an is-a hierarchy, subtle changes (*e.g.*, a change in model fragments localized low in the is-a hierarchy) can be distinguished to some degree from more extensive transformations (*i.e.*, a change in model fragments higher up in the is-a hierarchy). On the *path (segment) level*, intermediate model fragments may be abstracted when they are at the same level in the is-a hierarchy as the ones occurring in the begin and end of the path (segment). On

the *global simulation level*, an overview is possible of all situations and processes which can occur, highlighting the commonalities and differences between alternative paths.

Causal events: on the *state and transition level*, causal relationships are specified between quantities potentially influencing each other, or when they are proportionally related. However, since causal relationships may be inactive, and they may have opposing effects, their expected effect does not always occur. It's necessary to look at the *local level* to see which causal relationships did actually have an effect, and which were *submissive* (De Koning et al., 2000), *i.e.*, did not have an effect. On the *path segment level*, some of the local events can be connected to form a causal chain of events taking place *over multiple states*, *e.g.*, a temperature difference introduces a heat flow process, causing the temperature and pressure to rise in $state_m$, the pressure to reach its maximum in $state_n$, the container to explode in state and the fluid to leak out in $state_p$, the table to get wet in $state_q$, etc. Since a path segment does not include branching, we can say that the situation at the begin of the segment must lead to the situation at the end. On the *path level*, a path can include a branching point; in that case we can only say that the begin *may* lead (instead of *must* lead) to the end. On the *global level*, it's important to realize that the input to the simulator can lead to any of the end states via any of the possible paths, so we can only make causal statements about what all end states have in common.

3. Hierarchical abstraction of qualitative simulations

Now that all event types have been introduced, this section will describe their role in the abstraction process to facilitate the communication of the simulation results and underlying causal explanations. To relate our discussion more clearly with previous work, we have divided this section in two parts: aggregation of the state-transition graph, and aggregation of causal models.

3.1. Aggregation of the state-transition graph

The number of states generated in a simulation depends essentially on the scope and level of detail of the model: the number of independently varying quantities (responsible for branching), the number of qualitative distinctions in the quantity space of these variables, and the number of causal relationships included in the model. While any of these distinctions may be considered interesting for future users by the model builder, or may be necessary to calculate results further along a causal chain, not all of the distinctions matter at the time of presenting the results. The original state-transition (or behaviour) graph that results from a simulation can contain many (tens, hundreds or even thousands) behaviours, but as soon as the number is larger than a handful, it becomes difficult to gain an overview of what happens, especially when the state-transition graph contains branching. Reduction of the state-transition graph helps learners to gain an overview of the results, while parts of the graph may be selected by the user for further expansion, thereby giving access to the underlying details.

We distinguish three main methods of graph reduction: (1) abstracting from particular domain structures; (2) abstracting from particular kinds of events; (3) abstracting from temporal information. The first method, abstracting from particular domain structures, is very powerful, as shown by the following. Assume we're only interested in one of the three subsystems involved in the example simulation; now we can abstract away everything from our example simulation trace except the information pertaining to that subsystem. Suddenly, of the 19 original states, only 6 states remain, because the other states did not differ from the remaining states with respect to the subsystem of interest. Although we acknowledge the importance of this method, it has been treated in some detail by Mallory et al. (1996), and it requires (user) specification of interests. In this paper, we therefore focus largely on the second and third method of abstraction.

An overview of the top-level algorithm and the results of the different steps is shown in figure 3. We use two main principles behind each step in the algorithm: (1) we prefer linear descriptions of what happens, and abstract from alternative lines of events whenever possible; (2) we focus on begin and end of event sequences, if the intermediate stages are mostly continuous. In the following subsections, the specific techniques illustrated in figure 3 will be described in more detail. The example simulation that is used throughout this section is based on a re-implemented model of the Cerrado Succession Hypothesis model as originally presented in Salles & Bredeweg (1997).[2]

[2] Due to lack of space, and because we prefer to stress the generality of the approach, we do not go into more detail about the domain model in this paper.

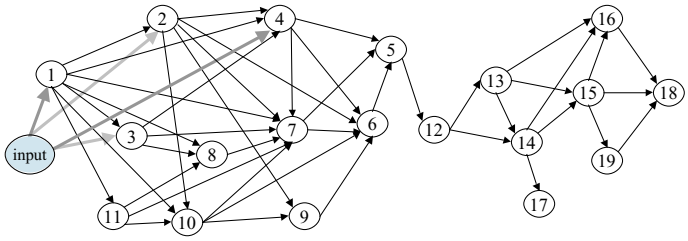
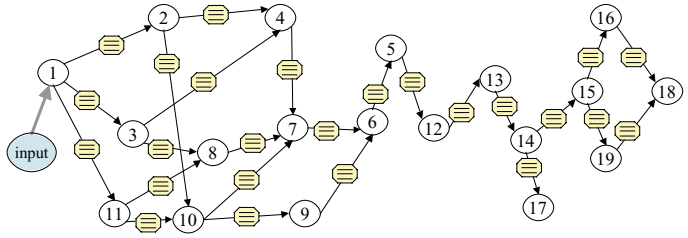
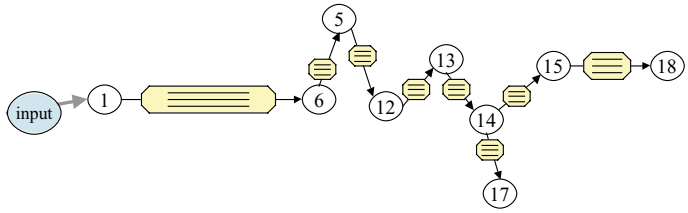
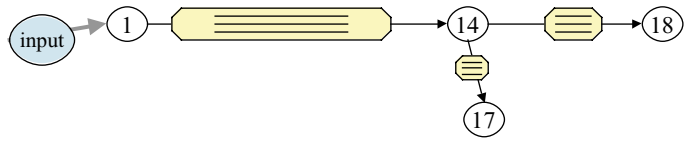
	The original state-transition graph as output from the simulator, consisting of 19 states, 43 transitions, and 896 behaviour paths. The first step in the aggregation process actually adds more information, namely the differences between states. The second step is performing *transitive reduction* (see text for details).
	The result after the first two steps, consisting of 19 states, 25 transitions, and only 24 distinct behaviours. The next step reduces the number of paths by *abstracting from alternative orderings*. This is done by comparing the events in reuniting branches, and creating higher level transitions when they match.
	The result after aggregation of alternative orderings, consisting of 9 states, 7 transitions (of which 2 are aggregated), and only 2 distinct behaviour paths. The final step, *aggregation of sequence*, compresses the events in a path until a branching point occurs, thereby reducing the number of states.
	The end result at the highest level of abstraction, consisting of only 4 states, and 3 aggregated path segments, which together summarize the two behaviours. Further abstraction would only leave the begin and end states.

Figure 3: Steps in the process of aggregating the state-transition graph

Transitive Reduction

Transitive reduction (as expressed by the first condition of algorithm 1) is a well-known technique in graph theory; it reduces the number of edges, while preserving all information, provided that the edge-relationship is transitive (*e.g.*, the hierarchical is-a relationship). In our case, the edges represent transitions which can also be considered transitive in some sense: the information that state 7 can be reached directly from 3 can be abstracted, because 7 can also be reached via some other path (*e.g.*, 3 → 4 → 7). There is an exception, however, when the events in the direct transition don't match the events in the longer path. In that case, the shortcut involves less, more, or different events than the longer path, and they should be considered as alternative behaviours. Therefore, the second condition is added to ensure that we only abstract away transitions in which the same events occur as in the longer path. The only information we lose after transitive reduction is that events may occur simultaneously.

Algorithm 1. Transitive reduction of the state-transition graph:

Abstract from (*i.e.*, remove) all transitions T (= X → Y) for which holds:

There is a path P from X to Y which does not contain transition T

AND

P contains the same events as T.

Since this technique involves looking at transitions, the events that should be considered in the comparison of P and T are at the local level. All events can be considered, or a subset of interest. Some events (like local maxima) may exist only in the longer path because they involve two transitions; in such cases, the second condition does not hold. Note: in this step, only transitions are abstracted, not states.

Aggregation of alternative orderings

The previous technique abstracted away the occurrence of events simultaneously, if they also occurred in sequence. We can generalize this idea of abstraction from sequence, by comparing the sets of events in different branches which reunite again later, *e.g.*, 15 → 16 → 18 and 15 → 19 → 18. If these different paths contain the same events (in a different order), or when the events can be aggregated to the same events, we can perform aggregation of alternative orderings, and see them as one *aggregated alternatives* transition, until the user is interested in more detail and the order becomes important again. The algorithm is presented here as algorithm 2.

Algorithm 2. Aggregation of alternative orderings in the state-transition graph:

Find a group of paths P_1 to P_n with the same begin-point (X) and end-point (Y), for which holds:

P_1 to P_n contain the same events, or events which can be abstracted into the same higher level events (following figure 1),

and do the following:

1. Add a shortcut edge from X to Y, to represent an aggregated transition, containing all (aggregated) events occurring in paths P_1 to P_n.
2. Delete every edge from the original paths P_1 to P_n, unless:
    a. the edge appears *after* an *incoming* branching point, OR
    b. the edge appears *before* an *outgoing* branching point.
3. Delete states which have no incoming and outcoming edges anymore.

Repeat this process (including step 1, 2 and 3) until no more alternative paths can be found. We assume that the procedure responsible for finding groups of equivalent paths starts with the shortest paths, so that the abstraction is done bottom-up. The *unless*-conditions in step 2 of the algorithm are necessary to prevent deletion of an edge when this would also cut off other paths than the ones abstracted.

Using this technique, both states and transitions are abstracted, thereby reducing the number of paths, or apparent ambiguities.

Aggregation of sequence

In this step of the aggregation process, path segments (sequences of states without branching points) are chunked into one *aggregated sequence* transition (the algorithm is straightforward, and omitted to save space). This technique further reduces the number of states and transitions, but not the number of paths.

3.2. Aggregation of causal models

With the term *causal model*, we mean essentially the set of causal relationships between quantities occurring in the simulation on the state level, but in a broader sense, also the causal relationships between higher level events.

On the state level, we have influences and proportionalities between pairs of quantities. Because the network consisting of these dependencies may be complex (involving tens to hundreds of relationships) it's useful to consider meaningful portions of it:

1. A quantity Qx (indirectly) influencing quantity Qy. Special cases of this include feedback loops, and/or mediating quantities;
2. A quantity Qx directly influencing all quantities Qy_1 to Qy_n;
3. All quantities Qx_1 to Qx_n directly influencing one quantity Qy.

These three cases enable highlighting of linear propagation of an influence, an influence spreading in multiple directions, and multiple influences combining, respectively. In combination, they can be used to explain why any quantity Qx is increasing, steady, or decreasing.

When besides the dependencies themselves, also the quantities' values and derivatives are considered, this creates more potential for abstraction, as demonstrated by the aggregation technique of De Koning et al. (2000). First of all, the status of each dependency can be labeled *dominant*, *submissive*, or *balanced*. This indicates whether their effect is as expected, is dominated by other effects, or balanced out, respectively. The distinction is used to abstract from all submissive dependencies, and focus only on the effects that lead to actual value events. Second, causal chains are constructed in which non-branching sequences are chunked, and fully

corresponding quantities (*i.e.*, which behave in exactly the same way) are grouped together as one. Third, the causal chains which do not directly lead to a state transition from the current state, are discarded.
The goal of De Koning's abstraction method was to facilitate hierarchical diagnosis of learners' reasoning, but we believe that this approach is also useful for explanation purposes. However, we propose the following changes to De Koning's abstraction method, two minor, and two more important points.
Leaving out submissive relationships simplifies things a lot, but we think this should only be done when a learner is already familiar with these relationships. Chunking sequences of relationships and grouping of fully corresponding quantities are both useful, too, but when the aggregated quantities belong to different entities, this may be a reason for keeping them separate. A more important point, however, regards discarding the causal chains which do not directly lead to a state transition. This is not desirable for explanation purposes, because it may disconnect an effect from its ultimate cause, as indicated by the following example. When an influence is introduced in $state_n$, this causes some amount Q (whose value currently lies in some interval, e.g., *low*) to increase. This increase does not directly lead to a state transition, however (e.g., because other quantities reach another qualitative value first), but it does so three states later, only then reaching the border of the interval *low*, and changing to *medium*. De Koning's mechanism would only include the causal chain from influence to the changing quantity in $state_{n+3}$, although the trend was already started in $state_n$. Instead, we propose that a causal chain is introduced as soon as the cause occurs, and that it is discarded only when it does not lead to *any* transition event later on in the simulation. As De Koning et al. note (2000), humans often make inferences and claims about events happening at some later point in time, not necessarily the first next state. Our suggestion addresses this concern.
The second significant change with respect to De Koning's mechanisms, is that we do not only include state transitions as events, but also other types of events, most notably derivative changes. This allows us to explain, on the local level, why a quantity Qx starts to increase, reaches a (local) maximum, or any other such type of event. Although we include some extra information with respect to De Koning's mechanism, we also allow further abstraction, by glossing over continuous developments. For example, in our view, it does not make much sense to explain why a quantity Qx *keeps* increasing, when the cause for it to start increasing has been explained already, as long as the same influences are applicable.

4. Discussion, Conclusion and Further Work

In our work we use qualitative simulations of system behaviour as interactive knowledge models. Such simulations are constructed by teachers, or with help of teachers, and capture insights that should be acquired by learners while interacting with these simulations. However, qualitative simulations include so much detail that learners may be overwhelmed by the amount of information. To fulfil the educational potential of qualitative reasoning in interactive learning environments, they need to be equipped with abstraction techniques to select the most interesting information from a qualitative simulation. To this end, we have presented a taxonomy of simulation events, and hierarchical aggregation methods to determine the most interesting behaviour of the simulated system. Our approach is more powerful than the work by De Koning et al. and Mallory et al., because it includes more types of events, and extends to aggregation levels above the local level to include path segments, paths and global views. It is less rigid than De Koning's $STAR^{light}$ system because, like Mallory's work, it transcends the low-level state-transition view to determine which events are interesting. It is also more flexible than Mallory's method because (like De Koning's methods) it does not require specification of user interests beforehand.

The algorithms described in this paper have all been implemented in SWI-Prolog (Wielemaker & Anjewierden, 1992). The visualisation of the aggregated results is currently being implemented as part of the model inspection tool VisiGarp (Bouwer & Bredeweg, 2001). Future work will focus on knowledge construction dialogues (e.g. Aleven et al., 2001) during which the learning environment takes the initiative and uses the hierarchically structured simulation model to actively support the learner in discovering the important behaviour features captured in the simulation.

References

Aleven, V., Popescu, O. and Koedinger, K.R. (2001). Towards Tutorial Dialog to Support Self-Explanation: Adding natural Language Understanding to a Cognitive Tutor. In: Artificial Intelligence in Education (AIED): in the Wired and Wireless Future. (eds) Moore, J.D., Luckhardt Redfield, R., and Johnson, L.J. pages 246-255, IOS-Press/Ohmsha, Japan, Osaka

Bouwer, A. and Bredeweg, B. (2001). VisiGarp: Graphical Representation of Qualitative Simulation Models. In J.D. Moore, G. Luckhardt Redfield, and J.L. Johnson (eds.), Artificial Intelligence in Education: AI-ED in the Wired and Wireless Future, pp. 294-305, IOS-Press/Ohmsha, Osaka, Japan.

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

Falkenhainer, B.C. & Forbus, K.D. (1991). Compositional Modeling: Finding the Right Model for the Job. Artificial Intelligence, 51, pp. 95-143.

Koning, K. de, Bredeweg, B., Breuker, J., and Wielinga, B. (2000), Model-based reasoning about learner behaviour. Artificial Intelligence, 117: pp. 173-229.

Mallory, R. S., & Porter, B. W., & Kuipers, B. J. (1996). Comprehending complex behavior graphs through abstraction. In Iwasaki, Y., and Farquhar, A., eds., Proceedings of the Tenth International Workshop on Qualitative Reasoning, 137–146. Menlo Park, CA, USA: AAAI Press.

Rickel, J., & Porter, B.W. (1997). Automated modeling of complex systems to answer prediction questions. Artificial Intelligence, 93, pp. 201–260.

Salles, P., & Bredeweg, B. (1997). Building Qualitative Models in Ecology. Proceedings of the International workshop on Qualitative Reasoning, QR'97. Istituto di Analisi Numerica C.N.R. Pavia, Italy, L. Ironi (ed.). pp. 155-164.

Wielemaker, J. & Anjewierden, A. (1992). Programming in PCE/Prolog. Dept. of Social Science Informatics, University of Amsterdam, The Netherlands.

Intelligent Agents an Approach to Supporting Multiple Model Based Training Systems

Keith Brown, Nick Taylor, Yanguo Jing, Tariq Khan*
Intelligent Systems Lab
Department of Computing & Eletrical Engineering
Heriot-Watt University, Edinburgh, UK
EH14 4AS
Tel. 00 44 131 451 3351
Fax 00 44 131 451 3327
keb@cee.hw.ac.uk
*Brunel University

Abstract

This paper presents an approach to using an intelligent agent architecture for a training system based on multiple models. In this architecture, the domain model contains all the domain knowledge of the target application; the user model contains all users' profiles. A methodology for organising the knowledge available is presented, along with an approach to explanations, which we believe are central to good training systems. The design of an intelligent agent based system for communicating this knowledge to various users in a relevant and context specific way is described.

Keywords

Multiple Models, Agents, Intelligent Training Systems, Explanations

Introduction

Within most industries a large variety of different kinds of knowledge are observable in the activities of different kinds of workers. The sum of all this knowledge may be described as expertise, which requires a wealth of different representations. An organisation's knowledge resource can be described in many ways, all of which should be integrated into accepted work environments, e.g. with simulations of industrial processes, to get maximum benefit for the entire organisation. Knowledge, so described, must be managed to prevent it from decaying or becoming stale. Management of knowledge is the process of understanding an organisation's intellect and ensuring measures are in place to maintain access to knowledge. This involves learning, which is the continual creation, evaluation, refinement and distribution of knowledge resources, and must be supported through explicit communication and interaction of the workers in an organisation. Agents that can tackle this problem are likely to form the backbone of many training and decision support systems in the future.

How knowledge may be described in a way that enables computer support for training and decision support is considered here. The context for describing knowledge is to support communication of this knowledge to workers through machine generated explanations. The communication process is essential in both decision support and training; the challenge is to provide access to the organisation's knowledge resources with relevance to a particular situation that a worker is working in. Context is essential here, for it dictates the demands on a worker's abilities and skills and therefore the usefulness of the information retrieved.

The activities of a worker in industries are varied and can range from straightforward routines to highly complex and cognitively demanding unique situations (Khan et al, 1997). Demands on a decision support and training system, therefore, involve requirements to describe all of the following knowledge sources: standard procedures, rules; basic principles, exemplars; instrumentation, technology. There are close relationships between work in artificial intelligence on knowledge-based systems (KBS) and modelling and work on information retrieval (IR) and databases. Whereas KBS are used mainly to solve problems, databases and IR systems are there to be a repository of knowledge. The position is taken in this project that that knowledge-based techniques and

knowledge repositories together can be used effectively for constructing models of knowledge by following a multiple models methodology in which different aspects of an organisation's knowledge resources are described in separate models. These must be supported with simulations that provide a way to activate the static knowledge descriptions and perform expert reasoning. A modelling methodology has been used to provide the structure for co-ordinating the various knowledge sources, and a model switching strategy has been produced within the EXTRAS project (Khan et al 1997) to govern use of this knowledge. Full details of the modelling methodology are provided in Leitch et al (1995). Here we briefly describe the modelling methodology, which assumes that knowledge has been identified already in a useable format. This is extended with methods for identifying certain kinds of knowledge resources for different purposes. Activities associated with the normal operations of a process plant are focused on, and the procedural, associational and principles type knowledge are included. Activities related to project work, design, testing, analysis, customer support, personnel, team work, etc., all of which are equally important, are not modelled explicitly.

We will describe how a multiple models methodology serves an organisation's on-going processes and helps preserve and communicate its knowledge: multiple models can contribute to managing knowledge by *modelling, decision support* and *training*. We will define a typology (modelling dimensions) for describing knowledge. Personal experiences (heuristics) need to be recorded as well as professional knowledge (principles). Therefore, multiple ways to describe the different elements of knowledge are suitable. A *multiple models* approach to describing the well formularised abstract aspects of knowledge is presented.

The field of intelligent interface agents has emerged during the past few years to address the increasing complexity of current software systems and is particularly suited to support of Intelligent Training Systems. They can give timely, beneficial assistance to users by extracting and analysing relevant information from the application domain knowledge and a user's profile. These agents, which each have their own role, can keep the models up to date and control the interaction with the users. They are sometimes termed personal assistants. Some of the existing interface agents focus on a user model based on a user's interaction history [Maes, 1994][Lau, 1999][Farrell, 2000]; some focus on user intent ascription [Brown, 1998]; Höök [1996] and Bomsdorf [1996] try to decompose user tasks to make mappings between tasks and interfaces.

We propose a model based intelligent interface agent architecture, see Figure 1 to support interaction between trainees, target application domains and interface agents. In this architecture, the Facilitator maintains a knowledge base of the capabilities of a collection of agents. Agents, including the intelligent interface agent, the target application and other agents communicate with each other through this Facilitator. The other agents include a user model management agent, a domain model management agent and an explanation agent.

A domain model (DM) contains an explicit representation of the target application. The intelligent interface agent (IIA) gets information from the domain model; it infers operations and sends instructions to the target application. The domain model consists of a task model (TM), a procedural model (PM) and an associational model (AM). The task model structures tasks in an AND/OR graph. The procedural model is an executable representation of procedures, which are ordered sets of tasks or actions. The associational model links situations to tasks or operations. These three components work together to represent the domain model.

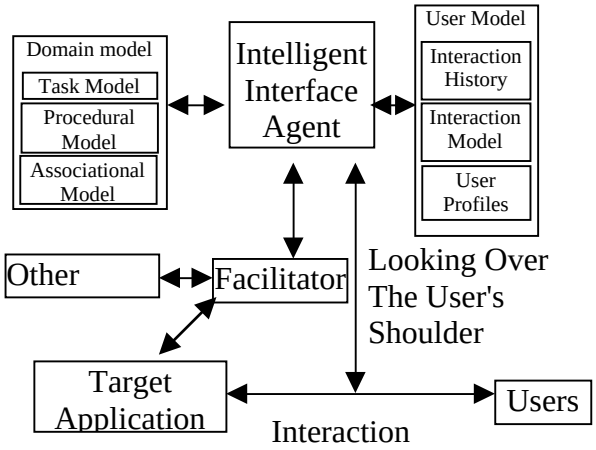


Figure 1: Model based intelligent interface agent architecture

In order to provide explanations for the users, the intelligent interface agent also need to get the user's personal profiles. The user model contains the user's information, which consists of an Interaction History (IH), an interaction model (IM) and a user profile (UP). The user management agent records in the interaction history the user's interaction with the target application. The interaction model represents a user's interaction habits with the target application, which is inferred from the interaction history. The user profile contains the static information about a user such as name, gender, knowledge level, etc. The intelligent interface agent (IIA) keeps track of the users' interaction with the target application; combines the knowledge from the domain model and the user model; decides upon assistance.

Modelling Methodology

Practical know-how and professional knowledge all need to be described and modelled in a multiple models architecture. The view of the modelling process used here consists of several modelling choices which involve the definition of many model properties. First, the *Ontological* choices reflect those aspects of knowledge being modelled, such as where the knowledge comes from. Next, the *Representational* choices dictate how the selected knowledge can be best represented. Also inherent in this dimension is *Generality* which specifies the level of abstraction of the knowledge. The representational factor suggests that there has to be three fundamental types of knowledge representation, to describe equations, rules and procedures. In addition to three representations, there are three kinds of inference mechanism. Finally, the *Behavioural* choices consider aspects of the variables of a model. The entire set of modelling dimensions allows a model to be described in an exact manner so that differences between models are apparent along significant aspects.

Knowledge of an application domain can be segregated on the basis of *scope* into system, subsystems and components. Within the same scope, taking into account a different number of parameters can create models having different resolutions. The resolution of a model is high if the number of parameters considered in the model is comparatively high, and is low if the number of parameters considered is less. Further, knowledge of theoretical principles for the domain are best represented using groups of mathematical equations; heuristics behind the operation of the system, subsystems and components are best represented as sets of IF-THEN rules; and operating procedures are best represented using a procedural form, e.g., Petri-nets (Taylor 1990). By adopting the various representational formalisms (equations, rules and procedures), models having different *generality* can be created. Equation models are considered to be the most general form of knowledge representation and procedures the least general as they are specific to situations. The efficiency of a knowledge representation can be maximised by adopting the appropriate representational formalism.

In addition to scope, resolution and generality, we propose *precision*, which allows precise, less-precise or least-precise behaviours to be generated from the models using precise, less-precise or least-precise values for parameters with suitable quantity spaces.

The separation of knowledge and its encoding using the various representational formalisms has culminated in a cubical framework, Figure 2, which is useful for organising the multiple models. Model switching involves switching between the models within this cubical framework based on a strategy that is suitable for explanation purposes. In most applications the framework would not be fully populated with cubes as some of the knowledge would not be available. Any algorithms used need to be able to accommodate misssing cubes.

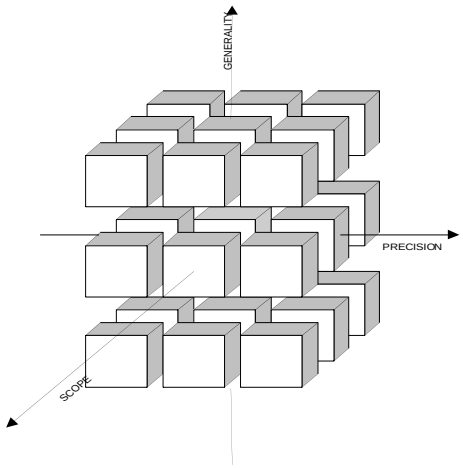


Figure.2: The model cube

Explanation using EXTRAS concepts for explanation agent

EXTRAS (Khan et al 1997) advances existing explanation systems by using many different descriptions of knowledge in generating dynamic explanations to recommend actions and justify practice. And by combining this with an on-line documentation facility that provides frequently requested information, a robust information retrieval capability is produced. The EXTRAS explanation strategy betters conventional approaches that just trace the system's reasoning, or vary presentation from text to graphics. These are insufficient for communicating the heterogeneous knowledge needed to support different situations encountered by operators of industrial plants. Changing presentation merely re-presents the *same* knowledge in a different way and fails to enrich the information content of an explanation. EXTRAS extends current assumptions of explanation generation by reducing explanation content to identifiable fragments of the corporate memory, and constructing explanations by switching between multiple domain models to satisfy the demands of different situations. This strategy needs a set of heterogeneous domain models from which to retrieve diverse kinds of knowledge descriptions. Multiple domain models are foundational in the EXTRAS project; without these, and a principled way of navigating the model space, it is not possible to generate meaningful explanations for the industrial systems under investigation. There are many benefits for explanation from using multiple descriptive models of knowledge.

Fundamental Considerations for a Domain System

Explanations can help communicate many kinds of knowledge, which include objective and subjective professional knowledge described as procedures, associations and principles. Procedures can be compiled from associations or can be based on basic principles. Often productions are used to represent associations in expert systems: 'If A Then B' type of rules define associations between preconditions and effects. Associations alone cannot give adequate didactic support, and causal relationships are better for tutoring because they provide scientific justification for the association (Clancey, 1987). Associations are often deduced from basic scientific principles, such as those for the laws of thermodynamics. One basic requirement for producing a domain system is to consider what kind of knowledge is to be provided by the explanation system. Another is to decide how to represent this knowledge. And a third consideration is how to present it (on-line documentation or dynamic generation).

To summarise, explanations can be provided at different levels of expressiveness. Each level clarifies tacit knowledge until the underlying principles for a domain are reached. Intelligent support should present all levels of explanation to satisfy the requirements of different users. This is achieved by describing different knowledge in suitable ways, i.e., procedures, rules and equations, although other representations, such as cases, are suitable for more informal knowledge. With an explanation strategy based on multiple domain models, the main requirement is to identify which model should be used to generate an explanation. That is, the problem is to decide which fragment of the domain system is relevant and useful for a given situation. Therefore, EXTRAS

provides a prescriptive explanation model that describes situations and maps them onto domain models. Situations are described using a refined representation of primitive tasks from the QUIC project (Leitch and Stefanini, 1988). A classification of explanations is produced based on the refined set of tasks, which allows access to different kinds of knowledge and domain models.

Classification of Explanations

A classification of explanations has been developed in EXTRAS. This classification is derived from well-established research in the theory of scientific explanation (e.g. Hempel, 1942/1948) and research in multiple modelling dimensions at Heriot-Watt University (e.g., Leitch *et al.*, 1994). Each explanation type has a distinct purpose that defines when that explanation is best. This is so that explanation types can be related to situations, independently of any domain specific descriptions.

The Structure of Explanations

There are several characteristics that an explanation possesses: design, extent, focus, substance and knowledge. These are examined below.

(1) ***design***: can be either didactic or informational.
(2) ***extent***: can be any of the primitive tasks, e.g., interpretation.
(3) ***focus***: can be any of the stages of problem solving.
(4) ***substance***: can be any of the hypotheses, e.g., causes, effects.
(5) ***knowledge***: can be any of the knowledge kinds, e.g., principles, associations.

Model Switching

Switching between different domain models is useful for dealing with uncertainty and ambiguity during training. It is realised that in a practical implementation, not all the elements of the domain system will be implemented, either for time and money constraints or because those aspects of knowledge are not available. Therefore, the model switching capabilities depends on the knowledge acquisition stage during which the models are implemented. Another factor, therefore, in designing the domain system, is deciding which kinds of model switching are suitable. A specific agent will be tasked with controlling the model switching.

Example1: If switching along the generality dimension is required, to alter the basis of problem solving, there must be models available that represent procedures, associations and principles for the domain. Without inclusion of these different kinds of knowledge and representations, the full pedagogical benefits of model switching cannot be obtained.

Example2: If switching along the scope dimension is required, to shift focus onto different parts of the system, there must be models available that represent different parts of the system. Any part of the system not represented in a model cannot be explained or simulated within the reasoning process. Separate models for each complete subsystem or component must be provided and indexed as a unique model. However, there remains the possibility of generating new models of comparatively complex systems from simpler atoms.

Example3: When the system is acting in a decision support mode, it must solve problems itself, and therefore needs to utilise the most appropriate domain model for the problem. Within an application, it might be decided to defaults to the procedure models level to find a procedure that can directly achieve a task goal. If no procedures are suitable then switching to the association's level to identify subgoals that can be achieved by available procedures is useful. Should this fail to find a procedure then equations models can be used to provide new facts for the situation, which help to better specify the problem. With the new facts, the advisory system will return to the associations level to apply new associations, and hence determine subgoals that can be achieved with available procedures. This entire sequence relies on each of the models being available, and so the application has the responsibility of ensuring that the models needed are available.

The intelligent interface agent gets all the knowledge of the target application from the domain model. The model translation into rules is quite straightforward [detail see Jing 2001a].

The User Model Design Method

The user model contains information about the user. It consists three parts - the Interaction History, the interaction model and the user profile. The interaction history records the user's interaction history with the target application. The interaction model represents a user's interaction habits inferred from the interaction history. The user profile contains the relatively static information about a user such as name, gender, knowledge level, etc. This information is the main knowledge for the user classification, we will mention this in the conclusion section. The user model management agent manages the user model including modify the interaction history, the interaction model and the user profile.

As a user interacts with a system, many events are generated and operations undertaken. All these events and operations constitute this user's interaction history. An interaction history consists of several sequences of operations, which are called action sequences. An action sequence is made up of a user's actions and system events. The interaction model is inferred from the interaction history to represents a user's interaction habits. At the moment, the interaction model inferring method takes into account the order of two actions (or events) in an operation sequence. There are two reasons, firstly, as the number of the actions whose order is taken into account increases, the computational effort increases exponentially. Secondly, the order between two actions can be used to infer the order among several (more than two) actions, as is shown in the prototype and experiments section.

Prototype and Experiments

We implemented this method in a system for age-related macular degeneration (ARMD) for diagnosis of eye disorders [Jing, 2001] using Java and Jess [Watson, 1997]. The main task of users of this system is to identify indicators of ARMD. A camera is used to obtain a raw image of a retina. Detailed information about the whole domain model content can be found in [Jing 2001a]

The user model management agent keeps track of the user's actions and the system's events, modifies the interaction history with new actions and new events. The interaction model inference is quite straightforward once the inference algorithms are implemented and the interaction history is provided. As an experiment, we record 102 action sequences of a user in the interaction history, using the IM inference engine we get the Interaction Model shown in Figure 3. This has demonstrated that the agent approach to support of a multiple model based training system is feasible. The different agents can be used for both decision support and training. In addition the idea of autonomous agents allows extra agents to be used for model maintenance and execution. The implementation of the agents to generate the explanations, which will be central to a good training system, remains to be done.

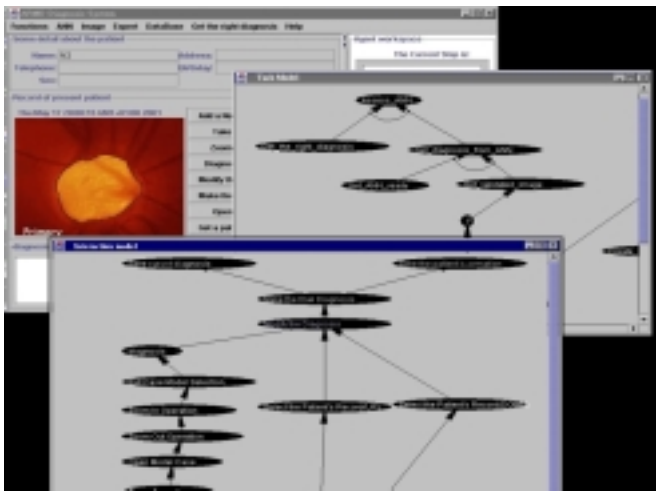


Figure 3 The ARMD System using model based intelligent interface agent architecture

Conclusions

In this paper, we have presented a multiple model based intelligent interface agent architecture to provide beneficial and timely assistance to trainees. A prototype system has been developed. The domain model and the user model have proved quite functional and helpful enabling the intelligent interface agent to give assistance to trainees. The methods presented in this paper produce good results according to our experiments. There are still several issues that need to be investigated.

References

Bomsdorf B, 1996, Christian Geiger, "Task as Agents: prototyping task models", *Proceedings of Sixth Australian Conference on Computer-Human Interaction*, IEEE Press, pp. 286-293

Brown S. M., 1998, *A decision theoretic approach for interface agent development*. PhD thesis, the Air Force Institute of Technology, Air University, USA.

Cawsey, A. (1993). Planning Interactive Explanations, *International Journal of Man-Machine Studies*, 38, 169-199.

Clancey, W.J. (1987). Knowledge-Based Tutoring: The GUIDON Program. The MIT Press.

Farrell R, Breimer E, 2000, "A task-based architecture for application-aware adjuncts", *Proceeding of International Conference of Intelligent User Interfaces'2000*, pp. 82 - 85.

Hempel, C.G. (1942). The Function of General Laws in History, *The Journal of* Hempel, C. G. (1948). Studies in the Logic of Explanation, *Philosophy of Science* 15 135-75. Reprinted in C.G. Hempel (1965). *Aspects of Scientific Explanation: and other essays in the philosophy of science*, The Free Press.

Höök K, Karlgren, Wærn, Dahlbäck, Jansson, Karlgren, and Lemaire, 1996, "A Glass-Box Approach to Adaptive Hypermedia", *In Int. Journal on User Modelling and User-Adaptive Interaction*, Vol. 9, pp. 157-184

Jing Y, 2001a, "The Domain Model Design in Model based ($\mathbf{MI}^2\mathbf{A}$) Architecture", Department of Computing & Electrical Engineering, Heriot-Watt University, *Technical Report, RM/01/4*.

Jing Y, 2001b, "The Interaction Model Construction method in Model based Intelligent Interface agent ($\mathbf{MI}^2\mathbf{A}$) Architecture", Department of Computing & Electrical Engineering, Heriot-Watt University, *Technical Report, RM/01/5*.

Khan, T.M., Brown, K.E. and Leitch, R.R. (1997). Didactic and Informational Explanation in Simulations with Multiple Models, in B du Boulay and R Mizoguchi (Eds.) *Artificial Intelligence in Education, Knowledge and Media in Learning Systems*, 355-363.

Lau TA, Weld DS, 1999, "Programming by demonstration: an inductive learning formulation", *Proceeding of International Conference of Intelligent User Interfaces'1999*, pp. 145-152

Leitch, R. and Stefanini, A. (1988). QUIC: a development environment for knowledge based systems in industrial automation, In *Proceedings of Fifth Annual ESPRIT Conference (ESPRIT 88*), ed. Commission of the European Communities, vol. 1, 674-696,

Leitch, R.R. *et al.* (1995). Modelling Choices in Intelligent Systems, AISB Quarterly No 93, 54-60.

Maes P, 1994, "Agents that reduce work and information overload", *CACM*, Vol. 37, No. 7, pp. 30-40

Taylor NK, 1990, “An Expert System To Assist Design”, PhD Thesis, University of Nottingham

Watson M, 1997, *Intelligent Java Applications for the Internet and Intranets*, Morgan Kaufmann, pp. 155-178

Object-oriented patterns for model-based reasoning

Tariq M Khan

School of Business and Management, Brunel University, UB8 3PH, UK.

tariq.khan@brunel.ac.uk

Abstract:

An attempt is being made to catalogue the key patterns that recur in the design of model-based reasoning systems so that reusability and good practice can be capitalised upon. Accordingly, some object-oriented patterns are presented here for the basic elements of models. These patterns cater for representing variables and parameters, both qualitative and quantitative, and associating them with the model elements they define. Further work will investigate relationships among the components of the patterns, e.g. dependencies between parameters.

Object-oriented patterns, model-based diagnosis.

Introduction

The development of models for use in model-based reasoning (MBR) is a laborious activity that could be simplified if models were made reusable. To this end, one of the most important recent developments in software engineering, *software patterns*, is considered here for potential application to model-based systems. Since the availability of the first collection of patterns (i.e. Gamma et al, 1995) the object-oriented community has worked to discover and record new patterns to capture common good practice in software development. Areas of application that have yielded patterns include design (Gamma et al, 1995), analysis (Fowler, 1997), architecture (Buschmann et al, 1997) and interface (Borchers, 2001). Although the exact form of a pattern differs in each of these areas the common thread is the motivation to provide a resource of proven solutions to common problems so that systems may be developed to accepted standards using common vocabulary. Reusability is made possible when developers are able to communicate their designs and systems to one another using familiar vocabulary and with confidence that reused components meet expected standards and work in expected ways. One of the most important benefits of patterns is that the learning curve is considerably reduced when

reusing existing components. A component that is claimed to implement a standard pattern can be understood relatively easily once its surface features are mapped to specific roles in the pattern.

This discussion paper has the intention to stimulate work in identifying and recording patterns that are applicable in MBR, and in particular, in education systems built upon MBR. A number of patterns are presented and analysed using object-oriented terminology and the unified modelling language (UML, Rumbaugh et al, 1999) as a notation. Only a few patterns can be presented here but there are potentially dozens of patterns that could be profitable for the MBR community. We take a didactic approach to the presentation following Fowler (1997) by building complex patterns from simpler ones, while discussing the limitations of the simpler patterns. The work is just begun as part of the CODIT project (EPSRC funded GR/R51346), which is examining model-based diagnosis in educational systems. It is expected that as the project progresses many more useful patterns will be identified and recorded for the MBR community. Some of these patterns may be original to the project and others may be adapted from previous applications in other domains, though they all will be presented with example of their use in dealing with core and recurring design issues in MBR.

Software Patterns

There is no agreed definition or format of a pattern so it is useful to provide a working definition first and then a format that can be easily understood.

Definition

"A pattern is a recurring organisation of objects that achieves required behaviour."

Format

In this paper a simplified format will be used for a pattern, which consists of a class association diagram in UML notation. It is common in the pattern's literature to provide additional textual description in a stylised form to include such categories as *Forces* and *Motivation* but that style is not immediately useful to this paper so it will not be adopted.

It is convenient to imagine patterns as syntactical structures in a language for communicating insight and experience about recurring problems and their solutions. Software developers should learn this

language of patterns to increase the quality of their designs by using available patterns wherever a suitable problem is recognised (i.e. one for which a pattern exists). The task of software design can be simplified significantly by referring to design patterns, both in the *generation* and *justification* of new designs. In other words, once design abstractions have been defined as patterns and given unique names, they can be called upon by name to conveniently specify a design and to convincingly justify the design decisions.

Examples

A few patterns will be presented next by describing their general form and function and how they relate to MBR. In places where reference to a specific application in MBR is instructive, the area of model-based diagnosis will be employed as a suitable illustration. The reasons for this choice of area are first, that cognitive diagnosis is the area of interest in the CODIT project (COGnitive Diagnosis in Intelligent Training), and second, that many of these patterns were developed from the analysis of the medical diagnosis domain and so are directly applicable to model-based diagnosis. The example patterns are deliberately chosen to deal with fundamental elements of a model (e.g. quantities and ranges) since they will recur often in any model and therefore offer considerable benefit in the process of model development.

The following patterns are developed from Fowler (1997) in which a series of patterns for Observations and measurements are described for the domain of medical diagnosis. The common theme is to provide ways of recording observations and measurements through such constructs as quantities, protocols and time records. The notion of a *type* is important and should be clearly understood in order to make sense of the patterns that follow. Simply stated, a type (abstract data type) is a structure that has operations which define its behaviour. The set of operations is known as the type's interface and is sufficient to distinguish between types. In object-oriented programming types are defined using interfaces (e.g. in Java) or abstract and concrete classes. Hence, often the terms type and class are used interchangeably but they are distinct since a class may correspond to several types. In essence a pattern is composed of several types that collaborate in a small part of the system (informally, a group of collaborating classes), hence the notation of class-association diagrams is appropriate for representing

many patterns (but not all!). In order to represent a modelling element, such as a quantity, a new type needs to be defined, which is the approach taken in the patterns described next.

Pattern 1: Quantity

The Quantity type is introduced to provide a flexible and meaningful way to represent measurements (e.g. temperature or exam result). Instead of merely creating a field called, e.g. “Exam Result” and associating a number to denote a student’s performance level, a new type is created that integrates both the numerical value of the field and the units of measurement, plus any operations that can be applied to the quantity. Note with this approach there is no difference in the way quantitative and qualitative quantities are treated in the model, however, a different representation for dealing with qualitative measurements is introduced later.

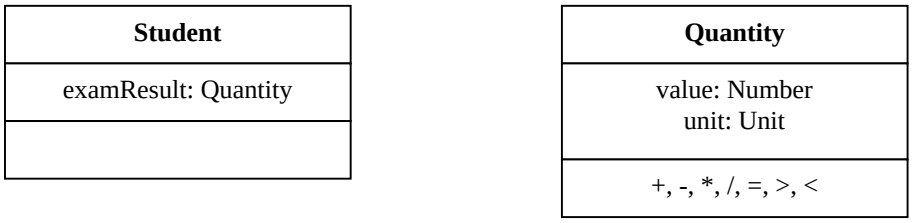


Figure 1: Two types: Student and Quantity

Figure 1 shows two simple types, Student, which represents the attributes of a student (e.g. examResult) that would be useful in a user profile, and Quantity, which is a general type that may be used in many situations to represent any quantity of interest. Quantity has two attributes, a value, which is a number (or a new type value for qualitative quantities) and a Unit, which is another type (defined below). The lower compartment shows the set of operations that are applicable on the quantity type. For example, the addition operator allows quantities to be added easily while ensuring that the integrity of dimensions is maintained (e.g. percentages are not added to absolute numbers). Where a conversion is needed, say from absolute to percentages, a new type called ConversionRatio can be added to qualify the Unit type, as shown in Figure 2 (operations are not shown).

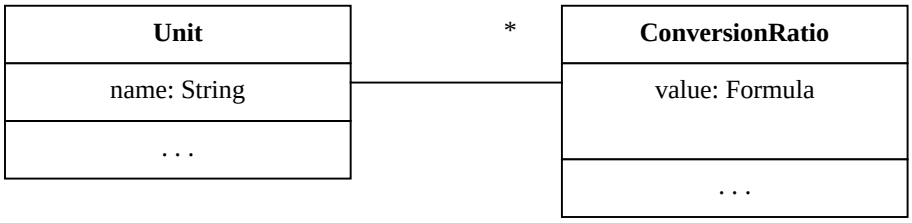


Figure 2: Conversion Ration added to Unit

Figure 2 shows an association between two types: Unit and ConversionRatio, which means that for any particular Unit, e.g. mile, there can be several ConversionRatios, e.g. ConvertToKm. How this conversion is performed is dependant on the formula in the ConversionRatio type (Formula would be another type).

Units can be subtyped in to atomic and compound units. For example, an atomic unit is Second (s) and a compound unit is Seconds squared (s^2). In this way compound units are constructed from collections of atomic units. Figure 3 below shows this revised pattern.

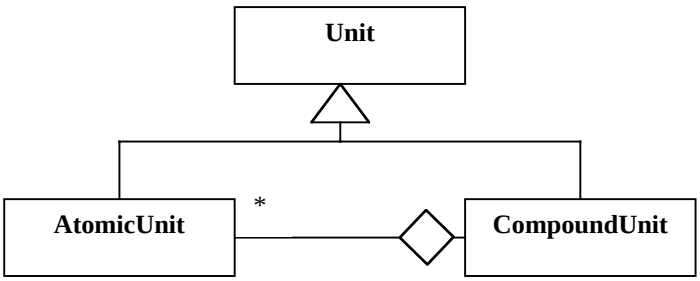


Figure 3: Generalisation structure for Unit types

Unit has two subtypes: AtomicUnit and CompoundUnit (shown as the triangle symbol to denote generalisation). A CompoundUnit may contain many AtomicUnits (shown as the diamond symbol to denote aggregation) and the relationships may be direct or inverse. For example, *area = metres squared* is a compound unit consisting of two direct atomic units (m X m). In contrast *velocity = metres per second* is a compound unit consisting of one direct atomic unit (m) and one inverse atomic unit (s^{-1}). Acceleration could be produced from either three atomic units, m, s^{-1}, s^{-1} or from one atomic unit plus one compound unit, s^{-1}, velocity. By adopting the approach of building compound units from other compound units, there is the benefit that all operations that are applicable to the aggregated units (e.g. velocity) will be available to the aggregate compound unit (e.g. acceleration), e.g. the conversion ratios associated with velocity.

One useful feature of the use of Quantity and Unit types is that multiple dimensions can be used for any quantity to be measured provided conversion ratios exist. When dealing with the problem of making sense of a student's answer to a probe question for diagnostic purposes, this flexibility in reasoning can be very beneficial since alternatives are easily handled.

Combining Figures 2 and 3 and adding the Quantity type we get the overall pattern for Quantity in Figure 4.

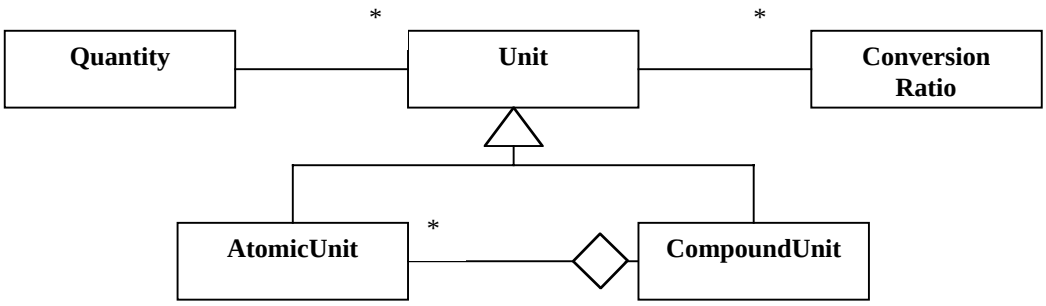


Figure 4: Overall Quantity pattern

Pattern 2: Quantitative

Although Quantity provides a flexible way to deal with individual measurements, a more powerful approach is to introduce a Quantitative type into which additional information may be added, such as when a measurement was made and under what circumstances. This is of particular interest in learner diagnosis since time is a crucial factor in verifying the applicability of any inferences made about the student's behaviour. Technically, the Quantitative type is an *association class*, since it represents the association between classes, Quantity, and two new types, Entity and Operand (plus additional types to represent time of measurement etc.). Figure 5 shows this arrangement.

*

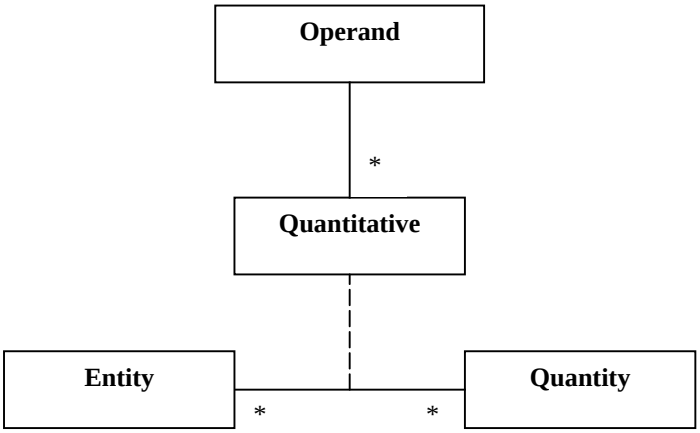


Figure 5: Quantitative and Operands

Entity represents the set of elements in the domain that possess measurable attributes (e.g. *Tank*, with attribute, *volume*). Operand represents the set of all measurable attributes of Entities in the domain (i.e. any variables and parameters).

Example 2.1: *Tariq Khan has achieved 78% on test 1* can be represented by a Quantitative object whose Entity is *Student* (Tariq Khan), Operand is *TestResult* and Quantity is *78%*.

Example 2.2: *Volume of water in tank B is 45 m^3* can be represented by a Quantitative object whose Entity is *Tank* (Tank B), Operand is *Volume* and Quantity is *45 m^3*.

Pattern 3: Observation

When dealing with quantitative measurements either the Quantity or Quantitative patterns are suitable, but in order to deal with qualitative measurements as well a new type is introduced called Observation. In fact, Observation is a supertype that envelopes both quantitative and qualitative measurements. Figure 6 presents the Observation pattern, which is a refinement of the Quantitative pattern.

In the Observation pattern a new type called Qualitative is introduced to cater for qualitative values, such as “big”, “fast”, “bright”. It is the parallel of Quantitative but has a Category type instead of a Quantity type. To denote this similarity a new supertype called Observation is created and used as the association class to associate *either* Entity with Quantity (for Quantitative) or Entity with Category (for Qualitative), but not both (indicated by the exclusive-or (xor) constraint). With this pattern it is possible to cater for both numerical and symbolic values in the model while treating them as distinct at

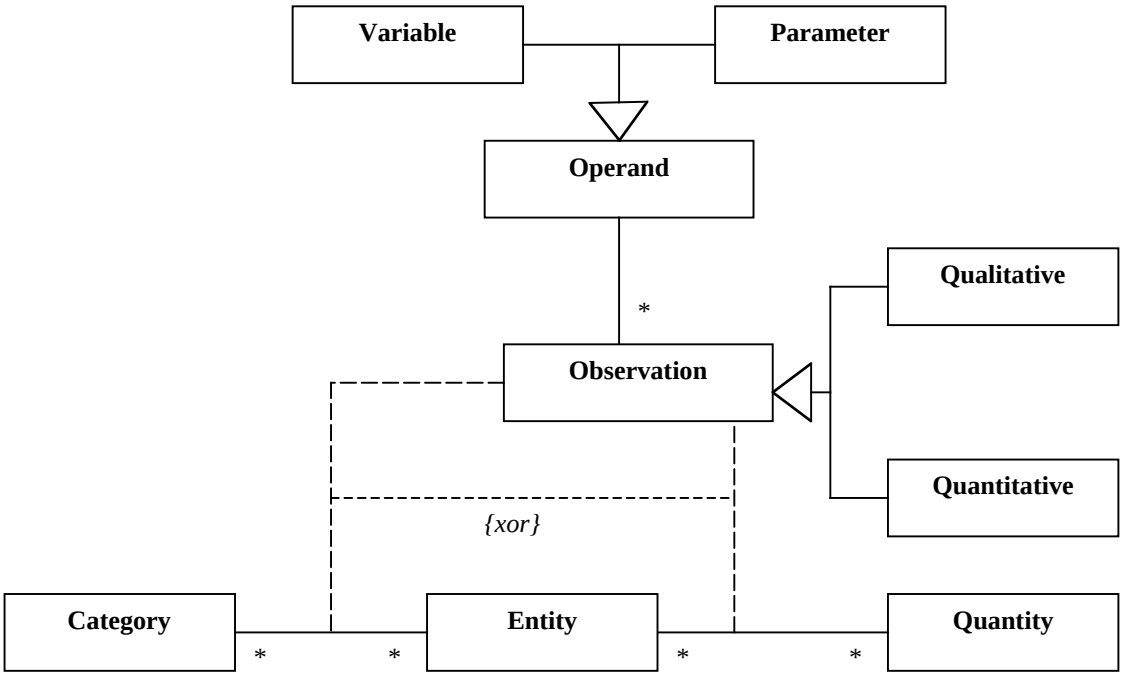


Figure 6: Observation pattern

one level of the model but similar at another level when considering only their supertype. Due to the benefits of polymorphism, which arise from the object-oriented programming paradigm, it is possible to ignore whether we are dealing with quantitative or qualitative data, at least at one level of abstraction. The Operand type has been subtyped in to Variable, which represents all directly measured attributes, and Parameter, which represents all calculated attributes.

Example 3.1: *Tariq Khan has performed well on test 1* can be represented as a Qualitative object with Entity *Student* (Tariq Khan), Operand *Test result* and Category *Well*. Compare this qualitative observation with the quantitative measurement of example 2.1.

Example 3.2: *Volume of water in tank B is half full* can be represented as a Qualitative with Entity *Tank* (Tank b), Operand *Volume* and Category *half full*. Compare this qualitative observation with the quantitative measurement of example 2.2.

We can develop the pattern of Figure 6 further to include the fact that categories are domain specific (e.g. “grade b”), whereas quantities are more general (e.g. $3ms^{-1}$). In order to describe a domain model the link between the Operand and its categories should be established by placing the association at a

permanent level, referred to as the *domain level*. If this were not done and remained at the transient *operational level* then the link would *only* exist once an observation was made (resulting in links between Category, Entity and Operand). The problem with this approach is that it could be inferred that the knowledge so described by the Observation exists *only* when an observation is made even though it should exist regardless of any observations. For example, a Operand *Exam* is graded by "grade A", "grade B" etc, which is a domain fact that should be stable in the domain model. It does not depend on there being observations of actual exams and their grades. Hence, it is productive to re-locate the type Category from the operational level to the domain level in order to establish a stable domain model.

A change in the pattern of Figure 6 is needed to reflect the relocation of Category. Figure 7 presents the modified pattern. Note that a supertype hierachical structure for Category may be introduced to enable generality. For instance, "grade A" and "grade B" are all kinds of "grades" and also kinds of "grade Pass". This structure is useful in diagnosis for it allows hierarchical reasoning, e.g. knowing that grade B was achieved, one knows that if it is a kind of grade Pass then it is reasonable to make the deduction that a grade Pass was achieved. The generality relationship supports reasoning by abstraction. Similarly, if a student possesses a bug about Kinetic Energy, then it may be deduced that they possess a bug about Energy via the generalisation relationship.

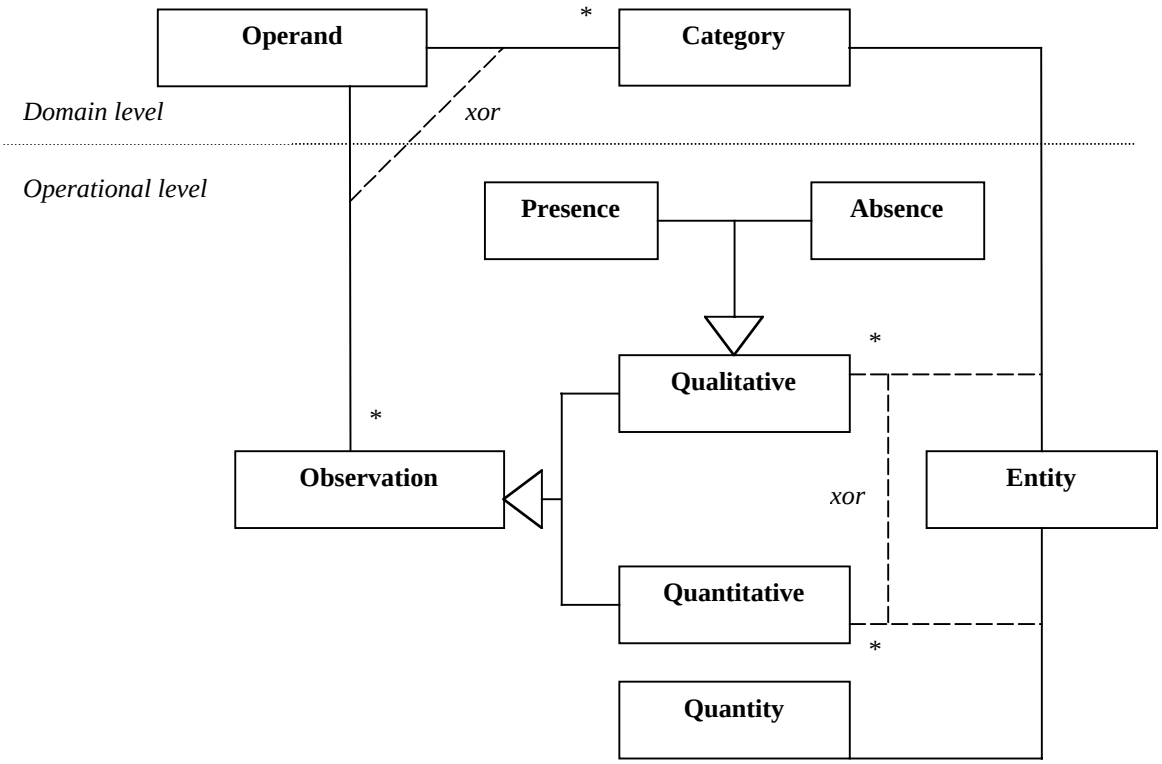


Figure 7: Revised observation pattern

Following this pattern, an observation may be a Quantitative that links to an Entity, a Quantity, to an Operand . Alternatively, an observation may be a Qualitative linked to an Entity and a Category, which in turn is linked to an Operand. The indirection inherent in the Qualitative means that the Operand is accessible only via the Category, whereas for Quantitative it can be accessed independently of the Quantity. Thus, emphasising the connection at the domain level between Operands and their possible qualitative values (Categories). There is no such strong coupling between Operands and their quantitative values (Quantities) since numbers have universal applicability.

Example 3.3: *Student Tariq Khan achieves 98% in test 1* is represented as a Quantitative object *Test-1*, linked to a Quantity object *98%*, Entity *Student* (Tariq Khan) and an Operand *test-result.*

Example 3.4: *Student Tariq Khan achieves a grade Pass in test 2* is represented as a Qualitative object *test-2*, linked to an Entity *Student* (Tariq Khan) and a Category, *grade-Pass*, through which there is a link to the Operand, *test-result.*

A further refinement in Figure 7 is the subtyping of Qualitative in to Absence and Presence, which is done to differentiate between the presence and absence of a Category. In diagnosis the absence of a Category can be as important as its presence, for it can help prune candidate hypotheses early. Compare a misconception (present Category) with a missingconception (absent Category). Introduction of the subtypes extends the range of observations that can be recorded in the model to include knowledge components that are known or are missing. Absent Quantitatives would not be useful since they would merely represent missing data, which would not allow any inferences to be drawn in the diagnosis. With respect to the generalisation hierarchy of Category, although presence of a subtype allows conclusions to be drawn about the presence of a supertype but not about the presence of a subtype, the absence of a subtype does not allow any conclusions to be made about the absence of a supertype, but only the absence of its subtypes.

Example 3.5: *Student Tariq Khan knows the correct relationship between pressure and temperature as a thermodynamic equation*, is represented as a Presence Qualitative, with Entity, *Student* (Tariq Khan), Category, *Thermodynamic-equation*, and Operand, *Correct-knowledge*.

Example 3.6: *Student Tariq Khan does not know the correct relationship between volume and temperature as a thermodynamic equation*, is represented as an Absence Qualitative, with Entity *Student* (Tariq Khan), Category, *Thermodynamic-equation*, and Operand, *Correct-knowledge*.

Example 3.7: *Student Tariq Khan knows an erroneous relationship between volume and temperature as a thermodynaic equation*, is represented as a Presence Qualitative, with Entity, *Student* (Tariq Khan), Category, *Thermodynamic-equation*, and Operand, *Erroneous-knowledge*.

In this example, Erroneous-knowledge and Correct-knowledge would be subtypes of Knowledge, which is the Operand. Other hierarchical structures are of course possible.

Conclusion

Some basic elements of models have been introduced and presented as object-oriented patterns. The motivation was to highlight the benefits of working with patterns so that a common standard for representing these elements emerges. Only the most basic of elements have been covered here and

there are very many others that need to be addressed. In particular, patterns are being developed for representing Ranges and Quantity spaces, which are considered essentially the same since they consist of intervals and points, with the difference being only the *type* of the point object (i.e. quantitative or qualitative observation). Furthermore, in object-oriented terms there is conceptually little difference between quantitative and qualitative observations so they may be treated equivalent in the domain model. Hence, operations and objects that interact with observations may take advantage of polymorphism as far as possible to generalise the model. Differences arise when implementation issues are introduced though and an appropriate pattern to represent this relationship should be used, e.g. the Bridge pattern (Gamma et al, 1995), which separates out the conceptual concerns from the implementation ones. It is hoped that some of these other patterns will be sufficiently developed for presentation and discussion at the workshop.

References

Borchers, J (2001). A pattern approach to interaction design. Wiley, ISBN 0-471-49828-9.

Buschmann, F; Meunier, R; Rohnert, H; Sommerlad, P and Stal, M (1996). Pattern-oriented software architecture, volume 1: a system of patterns. Wiley, ISBN 0-471-95869-7.

Fowler, M (1997). Analysis patterns. Addison Wesley, ISBN 0-201-89542-0.

Gamma, E; Helm, R; Johnson, R and Vlissides, J (1995). Design patterns eleents of reusable object-oriented software. Addison Wesley, ISBN 0-201-63361-2.

Rumbaugh, J; Jacobson, I and Booch G (1999). The unified modelling language reference manual. Addison Wesley, ISBN 0-201-30998-X.

Model-Based Reasoning for Domain Modeling, Explanation Generation and Animation in an ITS to help Students Learn C++

Amruth N. Kumar

Ramapo College of New Jersey
505 Ramapo Valley Road
Mahwah, NJ 07430-1680
amruth@ramapo.edu

We have been developing an Intelligent Tutoring System to teach students to analyze and debug C++ programs for semantic and run-time errors. In this tutor, we have used Model-Based Reasoning for domain modeling and explanation generation. In addition, we plan to use it for program animation. In this paper, we will present our design of the tutor, and results from evaluating one instance of the tutor in several sections of our *Computer Science II* course.

Keywords: Domain Modeling, Explanation Generation, Program Animation, Evaluation of Instructional Systems, Web-based Training Systems.

1. Introduction

We are developing an Intelligent Tutoring System to help students learn the C++ programming language by analyzing and debugging C++ code segments. Among the six levels of abstraction of educational objectives proposed by Bloom [3], we target **application** (use methods in new situations, solve problems using knowledge) in our ITS, as opposed to program **synthesis**, which has been the focus of many earlier works (e.g., LISP Tutor [13], PROUST [7], BRIDGE [4], ELM-ART [5] and Assert [2]). Our work focuses on tutoring programming constructs rather than the entire programming enterprise. It focuses on semantic and run-time errors in C++ programs as opposed to syntax errors that a compiler would detect.

We have been using Model-Based Reasoning [6] to model the domain for our tutoring system. In Model-Based Reasoning, a model of the domain is first constructed. In our case, this would be a model of the C++ language. This model is used to simulate the correct behavior of an artifact in the domain. In our case, the model is used to simulate the expected behavior of some particular C++ language construct, e.g., C++ pointers. This behavior is compared with the behavior predicted by the student for that artifact. The discrepancies between these two behaviors are used to hypothesize structural discrepancies in the mental model of the student for that artifact. In our case, the behavioral discrepancies are used to generate feedback to tutor the student.
Insofar as the domain model is complete, Model-based reasoning is comprehensive in its coverage of possible behavioral (and hence, structural) discrepancies. This is not necessarily true of Rule-Based systems (e.g., production rules used in ACT-R theory [1]), which cannot address behavioral discrepancies unless they have been explicitly encoded into the tutoring system. Similarly, Case-Based Reasoning systems are primarily constrained to the types of cases already entered into the knowledge base [14]

In the next section, we will discuss and analyze the benefits of using Model-Based Reasoning for domain modeling in a tutoring system. In Section 3, we will describe our domain model and how it is used to generate feedback in our tutor. In Section 4, we will describe the currently implemented features of the tutoring system, and present the results of evaluating the tutor in several sections of our Computer Science courses in Section 5.

Finally, we will discuss conclusions and future work in Section 6, including our plans to use model-based reasoning for program animation.

2. Model Based Reasoning for Domain Modeling

There are several advantages to using Model-Based Reasoning for domain modeling in an Intelligent Tutoring Systems to teach program debugging.

Domain Model is the Expert Module: We need not include the answers to problems in the ITS. The model knows the correct answer, i.e., it is capable of solving each problem to obtain the correct answer. Therefore, the domain model doubles as the runnable expert module. Constraint-Based Modeling [12] does not require the inclusion of a runnable expert module either. However, it targets the knowledge that prescribes user's actions whereas Model-Based reasoning targets the knowledge that describes the domain's behavior. In this sense, the two could co-exist in an Intelligent Tutoring System. There have been more recent efforts to extend Constraint Based Modeling to include a runnable expert module [9].

Dynamic Generation of Problems: Limited problem set has been recently recognized as a potential drawback of encoding a finite number of problems into a tutor [9]. Using Model-Based Reasoning for domain modeling can easily address this drawback. Since domain models based on Model-Based Reasoning are capable of solving problems on their own without being told the correct solution, a tutor using such models need not be restricted to administering only the problems that have been encoded into it. When coupled with a scheme for generating problems, such a tutor can potentially administer an unlimited number of problems to the learner.

One scheme used in literature to dynamically generate problems is by using BNF-like grammar, e.g., [8]. In this scheme, problems are generated by randomly instantiating the grammar. Each rule of grammar can be carefully designed with specific pedagogical objectives in mind. We have used such a generative scheme with our Model-Based tutor to be able to generate an unlimited number of problems. (Please See Figure 1, where templates are BNF-like grammar rules).

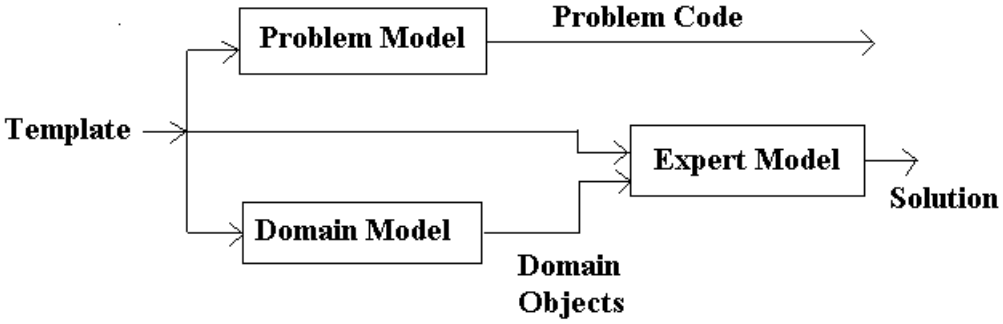


Figure 1: Generative Architecture of a Model-Based Tutor

Randomizing a template to generate problems is acceptable in the domain of program analysis and debugging, since the context of the problem is fully captured in the templates. However, this is not necessarily true in all the domains, e.g., in the legal domain [10], the context is much richer and harder to capture in templates.

One criticism that may be leveled at the process of randomizing a template is that it could generate only trivial variations of a "typical problem." In the programming domain, this is **not** true. Randomizing can yield sufficiently interesting and non-trivial variations of a problem. E.g., consider the following snippet of correct C++ code:

```
{
      int *variablePointer;
      int count = 32;
      variablePointer = &count;
      cout << *variablePointer;
}
```

One random variation of the template from which this code was generated is:

```
{
      int *variablePointer;
      {
            int count = 32;
            variablePointer = &count;
      }
      cout << *variablePointer;
}
```

Whereas the original code was correct, the new code suffers a *dangling pointer*, a rather sophisticated semantic error in C++ programming.

There are several benefits to coupling a tutor with a problem generator:

- The tutor is capable of administering problems on a particular topic to a student *as long as* the student needs, or *until* the student has mastered the topic.
- In addition to tutoring, the tutor may also be safely used to test students. Using a tutor to test students has many advantages: since students are already familiar with the interface of the tutor from tutoring sessions, they feel comfortable taking a test in this environment, with all the concomitant advantages of online testing. Since the tutor randomly generates problems, tests are individualized, deterring plagiarism.

3. Model-Based Reasoning for a Tutor on C++ Programming

We have been developing a tutor to help students learn C++ by analyzing and debugging C++ programs for semantic and run-time errors. We have used Model-based Reasoning for domain modeling in this tutor. In this section, we will describe our domain model and how it is used to generate feedback in our tutor.

3.1 Domain Model for Programming

The model of a domain consists of the structure and behavior of the domain. The structure describes the components in the domain and how they are interconnected and aggregated. The behavior consists of the relationships between inputs and outputs of the components and their aggregates.

The components in our domain model for the programming domain are program entities that have state, e.g., variables, constants, and functions. Each component is modeled in terms of the various states applicable to it, e.g., some of the states applicable to a variable include: declaration, allocation, assignment, and de-allocation. Unlike most other domains, we *do not* model the interconnections between these components – the interconnections are specified by control flow and data flow in the program. On the other hand, we model the hierarchy of composition of components, e.g., the models of all the variables declared in a function are composed within the model of the function. The hierarchy of composition of components reflects the scope hierarchy of entities in the program.

The behavior of each component is modeled as a state transition diagram. The state transitions are triggered by control and data flow in the program. Both valid and invalid states are included in the model. The invalid states of components correspond to syntax, semantic and run-time errors in the program.

The computation in a program is carried out in terms of expression evaluations and side-effects. In imperative languages such as C++, computation is primarily through side-effects. These side-effects are conveniently captured in terms of the state transitions of the components of a program model. Therefore, the simulation of a program model, corresponding to the execution of a program, is effected by transitions in the states of program entities. These transitions are themselves choreographed by the flow of control and data in the program.

3.2 Generation of Explanatory Feedback

Our tutor generates explanations about the behavior of a program when it simulates the model of the program. During the simulation, for each statement in the program, every component in the model that is affected by the statement contributes to the feedback as follows:

- **Process Explanation:** If the component undergoes a state transition during the execution of the statement, it appends an explanation of its state transition to the feedback.
- **State Explanation:** In addition, if the component reaches an invalid state as a result of the state transition, it appends a diagnostic message that it has reached an invalid state.

Aggregate objects (such as functions) collate the feedback statements contributed by their components (such as variables). The resulting generic explanation is post-processed based on the student model to produce feedback at various levels of granularity:

- **Simulative Feedback:** The feedback includes a complete explanation of the behavior of the program. This feedback is used for novices, for the first few problems in a tutoring session, and in instructional (as opposed to problem-solving) mode.
- **Diagnostic Feedback:** The feedback includes only the diagnostic lines of explanation that correspond to an error in the program. This feedback is used after the first few problems in a tutoring session, once the student starts making progress towards the educational objectives of the tutor;
- **Customized Feedback:** The feedback includes only those lines of explanation reported by the processes and objects flagged as being deficient in the student model.

To date, we have implemented simulative and diagnostic feedback in our tutor. The elegance of using Model-Based Reasoning for domain modeling is that the domain model naturally facilitates the generation of feedback, and no separate rules or constraints have to be encoded for this purpose.

One drawback of using Model-Based Reasoning for domain modeling is that building the domain model is an expensive task both in terms of time and expertise. However, once such a domain model is built, it will be able to handle any problem, and not just those previously encoded into the tutor. Unlike Rule-Based Reasoning or Case-Based Reasoning, Model-Based reasoning is not brittle. An implication of this is that the learner can enter his/her own problems into the tutor and test/learn from them, e.g., in Figure 1, we replace template by problem(s) entered by the learner. Such a facility in a tutor would be very powerful in promoting learning. The following table summarizes some of the ways a Model-Based tutor could be used:

Problems Generated By	**Problems Solved By**	**Type of Tutor Use**
Tutoring System	Learner	Tutor/Test Learner
Learner	Tutoring System	Solve for Learner
Tutoring System	Tutoring System	Demonstrate to Learner

4. Features of Our Tutor

Currently, our tutor for C++ programming knows about variables, scope, pointers, dynamic allocation and rudiments of function calls. We plan to develop several tutoring modules from this single domain model, addressing different aspects of C++ programming. One tutoring module that we have tested in several sections of Computer Science courses deals with pointers in C++. This module contains nearly 40 problem templates, and addresses dangling pointers and lost objects in C++.

Currently, the tutor provides four types of feedback:

- **None** – the tutor does not even indicate whether the learner's answer is correct. This is useful when the tutor is used for online testing in a class.
- **Demand** – the tutor provides feedback only on demand from the learner. The feedback provided may be minimal (states whether the learner's answer is correct or not), diagnostic (points out where the code has semantic errors) or simulative (explains the behavior of the code line by line, e.g., See Figure 2).
- **Error-Flag** – The tutor signals the correctness of the learner's answer by immediately changing the color of the learner's answer, red for incorrect and green for correct. The learner may follow-up by asking for feedback.
- **Immediate** – When the learner enters an incorrect answer, the tutor guides the learner through three levels of hints: **abstract** (e.g., "Remember, you have a dangling pointer if a pointer is dereferenced before it is assigned"), **concrete** (e.g., "Is valuePointer referenced before it is assigned?") and **bottom-out** (e.g., "Well, valuePointer has been assigned before it is referenced. Therefore, it is not a dangling pointer.").

The interface of the tutor consists of a left panel for the program, and a right panel for the problem, answering options and feedback provided to the user. The user is led through a clockwise flow of action that is intuitive:

from the program to the problem statement, answering options, grading button, feedback, and the button to generate the next problem (See Figure 2).

5. Evaluation of the Tutor

We have evaluated the tutor on C++ pointers in several sections of *Computer Science II* course. In this section, we will discuss the results of these evaluations.

Tutor in Isolation: In Fall 2000, we tested the tutor in two sections (N=19 combined), by administering a pretest (8 minutes), followed by practice using the tutor (10 minutes), and a post-test (8 minutes). These were not controlled tests. The author was the instructor in both the sections. The pretest and post-test scores were out of 40.

(N=19)	**Pre-Test**	**Post-Test**	**Effect Size**
Average	12.21	26.74	**2.16**
Std-Dev	6.70	8.73	

Note that **Effect Size is** *within-group*, i.e., (post-test score – pretest-score) / pretest-standard-deviation. An effect size of 2.16 indicates that the tutor facilitated learning among the students.

Tutor Versus Printed Workbook: In Spring 2001, we again tested the tutor in two sections (N=33 combined), using the pretest-practice-posttest protocol (8/10/8 minutes). We conducted a controlled test – between the tests, the control group practiced with printed workbooks, whereas the test group practiced with the tutor. The author was not the instructor in the sections. The pretest and post-test scores were out of 40.

(N=33)	**Pre-Test**	**Post-Test**	**Effect Size**
Test Group			**1.52**
Average	13.00	23.06	
Std-Dev	6.61	10.12	
Control Group			**1.33**
Average	15.24	24.71	
Std-Dev	7.10	10.54	

Practicing with the tutor appeared to be slightly better than practicing with the printed workbook, although the difference is not statistically significant.

Minimal Versus Simulative Demand Feedback in the Tutor: In Fall 2001, we conducted a controlled test of the tutor in one section (N=16). This time, we tested two versions of demand feedback for the tutor: minimal versus simulative. In minimal feedback, the tutor corrects the user's answer, but does not explain the correct answer. In simulative feedback, in addition, the tutor explains the correct answer. We used the same pretest-practice-posttest protocol as before, with fixed times for each step. Incorrect answers were penalized. The author was not the instructor in either class. The pretest and post-test scores were out of 80.

Section 1 (N=16)	**Pre-Test**	**Post-Test**	**Effect Size**
Test Group			**1.41**
Average	11.00	25.63	
Std-Dev	10.35	23.77	
Control Group			**1.63**
Average	8.25	17.38	
Std-Dev	5.60	14.62	

The results seem to indicate that simulative demand feedback may not be any better than minimal feedback. Informal comments from the students seemed to suggest that simulative feedback is too verbose. Research

indicates that to-the-point feedback is better than verbose feedback for promoting learning [16]. All the same, it is encouraging to see that using our tutor did help the students improve their performance.

6. Future Work

We plan to extend our domain model for program animation as follows:

- Each object will be in charge of its own visualization, especially local issues such as form and content.
- Aggregate objects will derive their visualization by composing the visualizations of component objects. Aggregate objects will be in charge of global issues for the visualization of component objects, such as layout.
- During simulation of the domain model, each object will render itself on to the visualization panel when it is first created in the program. After this, every event, i.e., action performed on the object will be animated through a corresponding change in its visualization.

There are several advantages in using model-based reasoning for program animation:

- There is no need to separately script the animation, as is the current practice (e.g., [11]). Since animation is entrusted to the model of the programming language, it is realized implicitly through message passing among the domain objects rather than explicitly through calls in the script.
- Since there is no need to write scripts to animate a program, the system can handle *any* program, without having to first preprocess it to generate the animation script. The completeness of the animation depends on the comprehensiveness of the model of the domain built into the system.
- Since the animation is assembled dynamically, this approach is more cost-effective than annotating programs individually.

For program animation, the actions performed during the simulation of the programs serve as adequate cues for model-based animation. On the other hand, for algorithm animation, the individual steps in an algorithm may not be at a sufficiently low level of abstraction to be directly translatable into cues for model-based animation. Therefore, model-based animation may be more suited for program animation than algorithm animation.

We plan to extend the Model-based domain model of the tutor to handle semantic and run-time errors associated with storage classes, arrays, structures, loops, nested selection statements, and their applications in C++. We plan to reify the tutor's interface by asking the user to not only choose the error in a program but also indicate the line of code where the user thinks the error has occurred. We also plan to continue to successively refine and evaluate the tutor in the future semesters.

Acknowledgements

This work was supported in part by a grant from the Ramapo College Foundation.

Partial support for this work was provided by the National Science Foundation's Course, Curriculum and Laboratory Improvement Program under grant DUE-0088864.

References

[1] Anderson, J.R. Production Systems and the ACT-R Theory. Rules of the Mind. Hillsdale, NJ: Lawrence Erlbaum & Associates, Inc., 1993, 1-10.

[2] P. Baffes and R. J. Mooney, A Novel Application of Theory Refinement to Student Modeling, Proceedings of the Thirteenth National Conference on Artificial Intelligence, pp. 403-408, Portland, OR, August, 1996.

[3] Bloom, B.S. and Krathwohl, D.R. Taxonomy of Educational Objectives: The Classification of Educational Goals, by a committee of college and university examiners. Handbook I: Cognitive Domain, New York, Longmans, Green, 1956.

[4] Bonar, J. and Cunningham, R., BRIDGE: Tutoring the programming process, in Intelligent tutoring systems: Lessons learned, J. Psotka, L. Massey, S. Mutter (Eds.), Lawrence Erlbaum Associates, Hillsdale, NJ, 1988.

[5] Brusilovsky, P., Schwarz, E. and Weber, G. ELM-ART: An intelligent tutoring system on the World Wide Web, in Proceedings of ITS 96 : Third International Conference on Intelligent Tutoring Systems, Montreal, Quebec, 12-14 June 1996.

[6] Davis, R. Diagnostic Reasoning Based on Structure and Behavior. Artificial Intelligence, 24 (1984) 347-410.

[7] Johnson, W.L., Intention-based diagnosis of novice programming errors, Morgan Kaufman, Palo Alto CA, 1986.

[8] Koffman, E.B. and Perry, J.M. A Model for Generative CAI and Concept Selection. International Journal of Man Machine Studies. 8 (1976): 397-410.

[9] Martin, B. and Mitrovic, A. Tailoring Feedback by Correcting Student Answers. Proceedings of Intelligent Tutoring Systems (ITS) 2000. G. Gauthier, C. Frasson and K. VanLehn (eds.). Springer (2000), 383-392.

[10] Muntjewerff, A.J. and Breuker, J.A. Evaluating PROSA, A System to Train Solving Legal Cases. Artificial Intelligence in Education: AI-ED in the Wired and Wireless Future. J.D. Moore, C.L. Redfield and W.L Johnson (ed.), IOS Press, Amsterdam. (2001): 278-285.

[11] Naps, T.L., Eagan, J.R.. and Norton, L.L. Jhave – An Environment to Actively Enhage Students in Web-Based Algorithm Visualizations. *Proceedings of the 31st SIGCSE Technical Symposium*, Austin, TX, March 2000, 109-113.

[12] Ohlsson, S. Constraint-based Student Modeling. In J.E. Greer, G. McCalla (eds.) Student Modeling: The Key to Individualized Knowledge-Based Instruction. (1994): 167-189.

[13] Reiser, B., Anderson, J. and Farrell, R., Dynamic student modeling in an intelligent tutor for LISP programming, in Proceedings of the Ninth International Joint Conference on Artificial Intelligence, A. Joshi (Ed.), Los Altos CA, 1985.

[14] Reyes, R.L. and Sison, R. A Case-Based Reasoning Approach to an Internet Agent-Based Tutoring System. Artificial Intelligence in Education: AI-ED in the Wired and Wireless Future. J.D. Moore, C.L. Redfield and W.L Johnson (ed.), IOS Press, Amsterdam. (2001): 122-129.

[15] Sack, W., Soloway, E. and Weingrad, P. From PROUST to CHIRON: ITS Design as Iterative Engineering: Intermediate Results are Important! In J.H. Larkin and R.W. Chabay (Eds.), Computer-Assisted Instruction and Intelligent Tutoring Systems: Shared Goals and Complementary Approaches. Lawrence Erlbaum Associates, Hillsdale, NJ, 1992, 239-274.

[16] Winkels, R. Explorations in Intelligent Tutoring and Help. IOS-Press, Amsterdam, Tokyo, 1992.

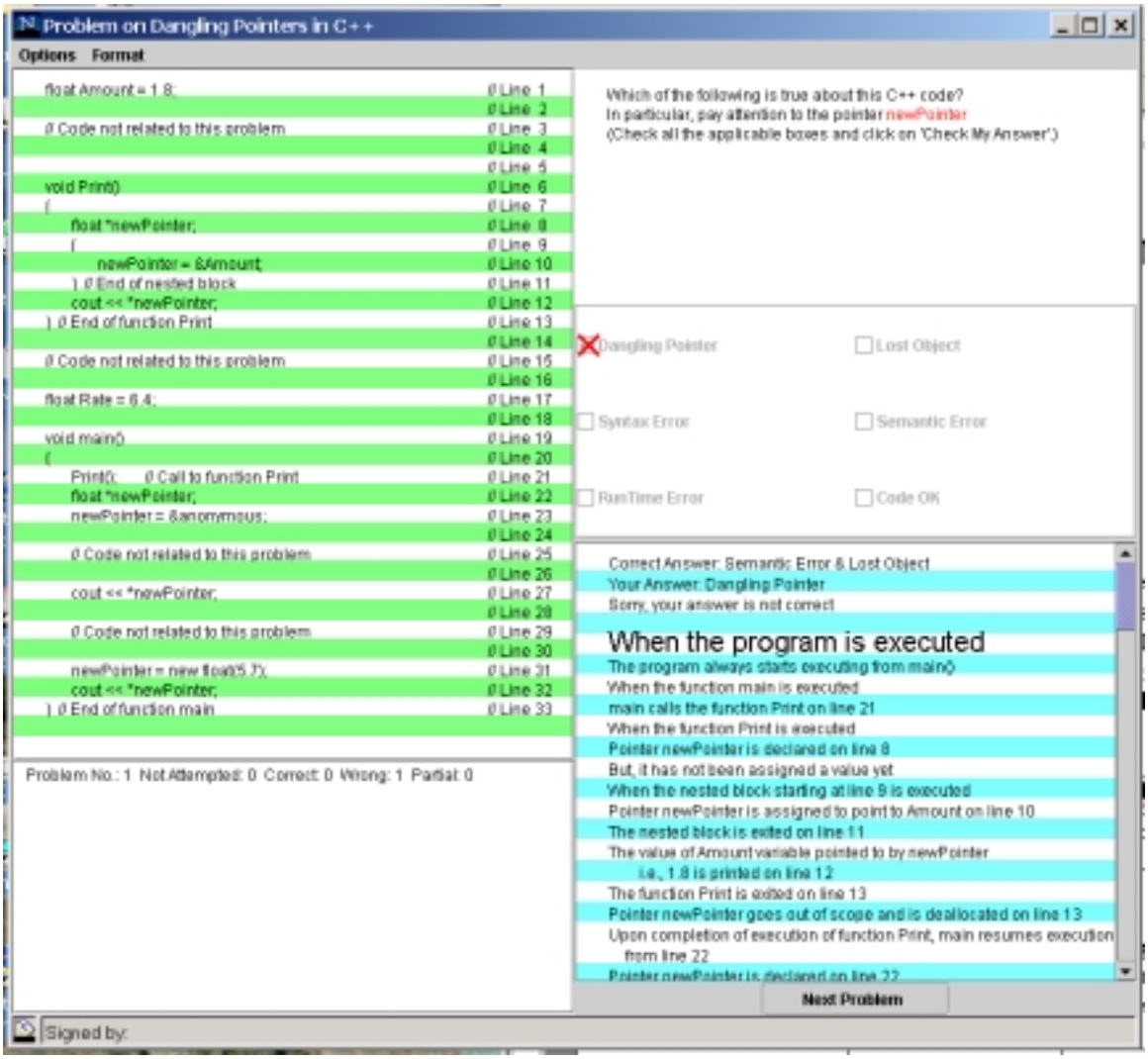


Figure 2: Screen Shot of the Tutor in action with Simulative Demand Feedback

Model-based Reasoning in Mathematical Tutoring Systems – Preliminary Report

Walther Neuper
Höhere Technische Bundeslehranstalt
Itzlinger Hauptstraße 30,
A-5020 Salzburg, Austria
neuper@cosy.sbg.ac.at

Franz Wotawa
Graz University of Technology
IICM – Software Technology
Inffeldgasse 16b, A-8010 Graz, Austria
wotawa@ist.tu-graz.ac.at

Abstract: This paper deals with the application of model-based reasoning to tutoring systems for teaching high-school-level mathematics. It presents the current state of such a tutoring system ISAC that has been developed for this purpose. The shortcomings of the current system and the question of how model-based reasoning can be used to overcome this drawbacks is discussed. In particular we introduce a framework for handling the knowledge that has to be dealt with for this purpose.
Keywords: Mathematical tutoring systems, Model-Based reasoning

1 Introduction

Model-based reasoning (MBR) is an old but still very fruitful and import area in artificial intelligence research. Nowadays MBR is used as a basic technology in several different application areas, including tutoring systems, software debugging, monitoring and control of technical system, mobile and autonomous systems, e.g., cars, space probes and robots. In the domain of tutoring systems Kees de Koning et al. [3, 4] have introduced a tutoring systems dealing with physics. The underlying MBR system is based on the traditional GDE approach [2] to model-based diagnosis [13].
In this paper we try to present another approach that should be suitable for mathematical tutoring systems that are intended to be used for supporting to teach students at the high-school level. Although, some of the basic ideas can be taken from Kees de Koning et al. there are some differences because of the different nature of both domains. Whereas physics is dealing with the relationship of quantities and causalities occurring in the real world, we have to deal with mathematical formulae and the mathematical calculus. Hence, means for representing the way of mathematical thinking is required.
Because of the intended application area, i.e., teaching math at high-school, we consider problems of applied mathematics, i.e. problems which apply knowledge given in a knowledge base (and do not consider the process of extending such a knowledge base). Problems are usually described by a text and figures. They have to be translated into a formal model (phase of *modeling*). The models itself are related to the respective notions of mathematics. These notations depend on the problem and require specific methods to solve the problem (phase of *specifying*). The specified notations of the problem solving methods and related knowledge can be seen as the background knowledge of mathematics, or the general mathematical knowledge. This kind of knowledge is different to the formal model of an example which can be seen as the example-specific knowledge. A good knowledge-based design would emphasize that the general mathematical knowledge is in a shape that it can be used to solve a huge class of examples. After modeling and specifying the problem can

be solved by constructing the result(s) (phase of *solving*). We further require the considered problems to be decidable, i.e., if the given values are within a certain range specified by a particular problem.
This paper takes the results of a mathematical tutoring system ISAC [11], identifies weaknesses of the underlying concepts, and discusses a promising solution to the open problems. ISAC is a generalized algebra systems which works stepwise and interactively. As soon as modeling and specifying have finished successfully, the specified method leads to the result. ISAC's method interpreter guides the user nicely step by step through the phase of solving. However, guidance in the modeling phase is still an issue.
The paper is organized as follows. In the next section we discuss the current implementation and concept of the ISAC system. We then show some open problems regarding ISAC. Having the problems in mind we introduce a formal framework that should be capable to solve the problems. The framework is based on MBR. Afterwards we discuss open issues and future research direction.

2 The ISAC system

2.1 ISAC's current way of problem-solving

We explain ISAC's problem-solving capacity by an example. Problem-solving capacity means (1) ISAC's capacity of solving a problem automatically, and (2) ISAC's capacity to generate explanations automatically. ISAC's user is called 'student'. The example belongs to a problem concerning the application of calculus, a problem which usually is being exercised in dozens of examples.

2.1.1 The phase of modeling

The following **description** is an exact copy from a text book [5].

> *A coil with a circle-shaped section and radius R should get a cross-shaped kernel (two equal bars with length v and width u) of iron, see Fig.1. Determine u and v such that the area A of the kernels section is maximal for a given R.*

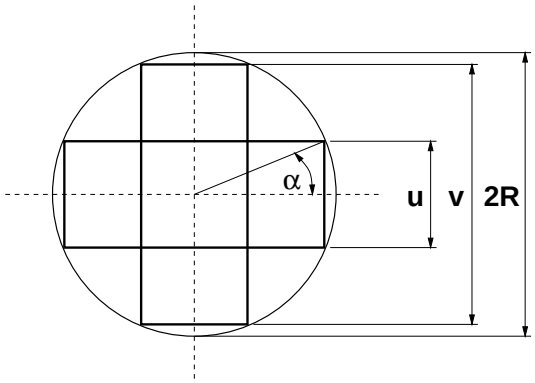


Figure 1: **Coil with a cross-shaped kernel to be maximized**

This description concerns notions of elementary geometry (*circle, radius, cross,* etc.) the student has to use in order to get a formal model of the problem. Such a formal model finally could look like

given : *Constants* $\boxed{R = arbFix}$
where : $0 \leq R$

find : *Maximum* $\boxed{A}$
AdditionalValues $\boxed{[u,v]}$
relations : $\boxed{A = 2uv - u^2,\ (\frac{u}{2})^2 + (\frac{v}{2})^2 = R^2}$

where the boxed formulae have to be input by the student, while the other information is provided by the system as a kind of template for this problem. The kind of model above [1] has been introduced by 'Formal Methods' in software engineering, see e.g. [8].

Generation of explanations is only possible in the modeling phase, if some author has prepared a **formalization** (in this example eventually with three variants F_I, F_{II}, F_{III})

$$\begin{array}{rcl} F_I & \equiv & (\ \{R = arbFix\},\ \ \{A, [u,v]\}, \\ & & \{0 \leq \frac{u}{2} \leq R,\ \ \{A = 2uv - u^2,\ (\frac{u}{2})^2 + (\frac{v}{2})^2 = R^2\}\ ) \\ F_{II} & \equiv & (\ \{R = arbFix\},\ \ \{A, [u,v]\}, \\ & & \{0 \leq \frac{v}{2} \leq R,\ \ \{A = 2uv - u^2,\ (\frac{u}{2})^2 + (\frac{v}{2})^2 = R^2\}\ ) \\ F_{III} & \equiv & (\ \{R = arbFix\},\ \ \{A, [u,v]\}, \\ & & \{0 \leq \alpha \leq \frac{\pi}{2},\ \{A = 2uv - u^2, \frac{u}{2} = R\sin\alpha, \frac{v}{2} = R\cos\alpha\}) \end{array}$$

and a **specification**

$$(real\_numbers, [optimization, by\_calculus], max\_by\_calculus)$$

i.e. a triple consisting of a **domain**, a **problem** and a **method**.
In general the formalization and the specification are hidden from the user. This hidden information allows the system to provide a minimum of help: The problem $[optimization, by\_calculus]$ provides for the template, and the formalizations allow to reject an input as 'unknown'. There are not explanations mentioning knowledge about the elementary geometry involved.

2.1.2 The phase of specifying

This phase makes the domain, the problem and the method explicit. These three kinds of knowledge can be seen as the axes in the **3D-universe of math** [1] which could look like Fig.2.
W.r.t. our example the domain *real numbers* contains the definition of $+, -, \cdot$ etc. together with the respective algebraic laws. Actually, the knowledge of the domain was already necessary for parsing the formulae in the modeling phase – without the hidden specification first of all the student has to specify a domain.
The axis of the problems is a hierarchy, which is indicated by the [ ] in the identifier of a problem: it is a list of keys pointing into the hierarchy, $[optimization]$ is the parent of $[optimization, by\_calculus]$. The latter **matches** the above model of the example, whereas say $[optimization, linear]$ would not match the model. The student can select a problem in the hierarchy, and get feedback if the model matches the problem. This kind of matching is described in [11].
In ISAC the axis of methods is still closely related to the problems (and not that independent as shown in Fig.2): each problem is linked to the solving method(s). Each method has a guard, which also has to match the model in order to avoid inapproriate application. The method solving our example is described in the subsequent section.

[1] The model does not presented the post-condition to student, rather some of the respective sub-terms in the field 'relations'.

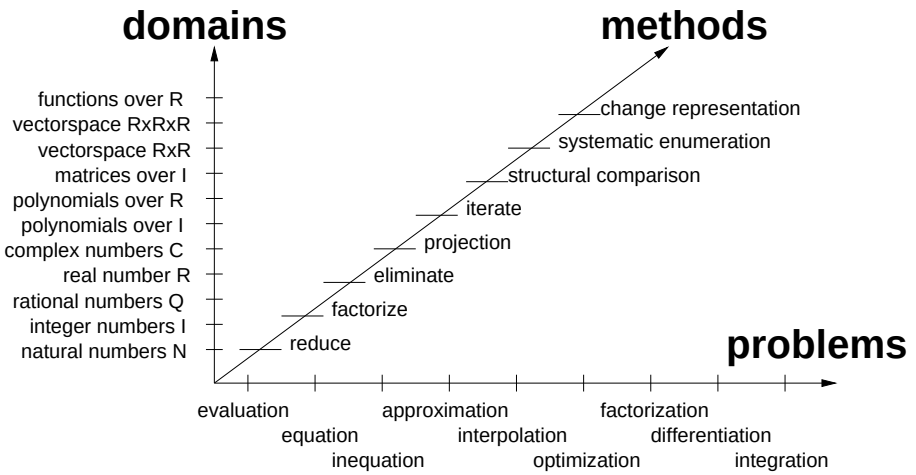


Figure 2: **The three-dimensional universe of mathematics**

Generation of explanations is better elaborated in this second phase of problem solving; the systems feedback is manyfold: due to a 'too weak' domain the formulae may not be parsed, during matching the model with a problem an item may be *missing* or *superfluous*, a pre-condition (in the field 'where') may be *false*, etc.
One remarkable feature of ISAC s problem hierarchy is, that it allows to **refine** the problem w.r.t. a model [11]. This feature makes the mechanism explicit which determines for instance the appropriate type of an equation input to an algebra system, and then applies the according method in order to solve the equation. The mechanism, made explicit in ISAC, thus generates structure related explanations.
Such explanations are given by **reflection**: ISAC just shows its own mechanisms (working on litterally the same knowledge the user is inspecting) and how they relate the concrete problem at hand to the general knowledge.

2.1.3 The phase of solving

The phase of solving is entered, if a model matches the guard of a method – thus prohibiting the application of an inappropriate method.
In this phase the result is acchieved by tactics stepwise constructing a proofstate (consisting of a prooftree etc.); the proofstate prohibits to apply tactics in an inconsistent way. The tactics belonging to a method are described in a **script**. An important tactic is **rewriting** the current formula by application of a theorem – this is the basic mechanism for algebraic formula manipulation. Again, a theorem not appropriate for the current formula is not being applied.
The student may input a tactic or a formula. The method interpreter locates a tactic, input by the student, within the text of the method, and proposes the next applicable tactic. If the student inputs a formula, the script interpreter locates the tactic which would have calculated this formula (eventually following several intermediate tactics). This mechanism is covered by a layer for dialog-guidance, which filters and varies the information transferred to the student (in order not to provide too much help, and not to become boring) [12].
Althoug ISAC prohibits inconsisten application of tactics, the student may apply tactics in a misleading way – leading to somewhere, but not to the result.
The script of the method *max_by_calculus* solving the problem $[optimization, by\_calculus]$ is as follows:

```
Script max_by_calculus (fix_::bool list) (m_::real) (rs_::bool list)
                       (v_::real) (itv_::real set) (err_::bool) =
```

```
(let
    e_ = (hd o (filter (Testvar m_))) rs_;
    t_ = (if #1 < Length rs_
        then (Subproblem (Reals,["make_fun"],no_met) [m_, v_, rs_])
        else (hd rs_));
    mx_ = Subproblem (Reals,["function","max_of","on_interval"],
            maximum_on_interval) [ t_, v_, itv_ ]
in (Subproblem (Reals,["tool","find_values"],find_values)
            [ mx_, (Rhs t_), v_, m_, (dropWhile (ident e_), rs_)]))
```

The task of this method is to compute and to transfer values to and from three *Subproblem*s. This means, that the phases of modeling and specifying may occur recursively.
The above method prescribes the sequence of steps. The general case in mathematical problem solving is, that several steps are possible in parallel. ISAC models this requirement by quasi-parallel evaluation of parallel expressions in the (functional [9]) description of methods, and by employing complete (in the sense of [10]) rulesets in rewriting, where the sequence of rule application is arbitrary.

Generation of explanations relies on reflection again. In the phase of solving the students requests for explanations are handled by ISAC, as if the student would have said 'give me the details'.
Details might be: Some more intermediate step in algebraic transformations (e.g. if a whole set of rewrite rules has been applied), the assumptions generated during a calculation, a (true or false) condition in conditional rewriting, an animation of matching etc. On the other hand the student has means to get a survey over the whole solving phase: he works on a sheet with a hierarchical structure, where he can fold in deeper levels of detail.
If a formula is input, the method-interpreter cannot calculate, the student is asked for the tactics leading to this formula.

3 Open problems of the ISAC system

Although the ISAC system is capable to help students understanding the problems of mathematics in an interactive fashion there are some problems with the underlying approach and its capabilities. Currently ISAC complains on student's faults as early as possible, rejects them by reasons of inconsistency of the proofstate, and comments them with rather technical explanations ('tactic is not applicable', 'formula does not match' etc.). 'Real world problems' in mathematics education, however, require much more additional knowledge (e.g. about elementary geometry) in order to solve them. No help can be given in this respect, and ISAC has no notion about what a human teacher would identify as a 'misconception'.
A way to solve the problem and to relax the underlying assumptions of ISAC would be to allow the student to go through an example as he or she would like. For example, it should be possible that the student specifies the mathematical problem and provides a solution of the problem to the system (which is different from going step by step through the solution process). The math tutor should then check the solution and in case of errors should report parts of the solution process that are responsible for the erroneous parts of the solution. Such a system behavior requires means for representing the expected cognitive solution process of the student for a given example. Because of the fact the the real cognitive aspects for solving a mathematical problem are hidden (in the brain of the student), finding a sequence of actions which the student might have in mind when solving the given example is in principle an unsolvable problem. An approximated solution would be to generate a formal representation of required actions, i.e., mathematical principles, that are required in order to come-up with a correct solution. Once the example is not solved accordingly

the system has to search for the principles that are violated. This approximated solution is sufficient for our purpose. Since we are interested in helping the student to use the correct mathematical principles for solving examples. Such an approach is very similar to the one proposed by Kees de Koning et al. [3] and more recently [4].
A particular problem of ISAC is due do its limitation regarding the problem formalization. The mathematical problem has to be specified in a formal way. Hence, the way from the textual representation of the problem that is usually given to the student and its formal representation is currently not supported by ISAC. For many students this is exactly the problem. Therefore, a system that helps the student to come-up with a formal description, once a textual description is given, is required. The key to a solution of this problem is again a more appropriate representation of the knowledge. The required steps are only loosely coupled with mathematical principles. This is different from our first situation where the mathematical principles (the teacher or tutoring system wants to teach) are one-to-one related to the steps of the solving process.
In the following section we introduce a framework that allows to specify the parts of the cognitive process which are required to construct a mathematical model for a given textual description.

4 A model-based reasoning framework

The basic element of the model-based reasoning framework is a *mathematical concept*. A mathematical concept can be seen as a function that alters the current state only if some pre-conditions are valid in the state. After the execution the current state becomes the previous state and the new state becomes the current state. This process stops until the pre-specified goal state is reached. Hence, the mathematical cognitive process, e.g., find a formal model for a given textual problem description, is given by an initial state, a set of mathematical concepts, and one or more final states. Formally, we defined the framework as follows:

Definition 4.1 (Mathematical Cognitive Process) *A mathematical cognitive process (MCP) is a tuple (I, M, G) where I is the initial state, i.e., a set of valid logical sentences, M is a set of mathematical concepts, and G is a set of goal states.*

Definition 4.2 (Mathematical Concept) *A mathematical concept is a tuple (N, P, A, C) where N is the identifier, P is a set of pre-conditions, C is a set of post-conditions, and A is a an action that describes the behavior of the concept. An action can be composed of other actions and may refer to other concepts.*

We define actions in a recursive manner:

1. A logical sentence (that may refer to a mathematical concept) is an action (basic action).
2. If $a_1, \ldots, a_k$ are basic actions, then
   (a) the parallel execution $a_1 || \ldots || a_k$,
   (b) the sequential execution $a_1 \oplus \ldots \oplus a_k$, and
   (c) the alternative execution $a_1 | \ldots | a_k$

   are actions.

This definition restricts compositional actions to be either actions that are executed in parallel or actions that are executed in a sequential order. A mixture of actions is not allowed for mathematical concepts. This restrictions is due to simplicity and does not restrict the whole conceptual framework. If a mixture is required, the different parts must be hidden in new mathematical concepts.

The above framework is similar to STRIPS planning [6] and Qualitative Process Theory (QPT) [7]. In contrast to planning, the notation of mathematical concept is not equivalent to actions in planning. In our framework mathematical concepts can be comprised of other concepts where there is no correspondence in STRIPS planning. The difference between our framework and QPT lies more in the nature of the domain of usage. QPT is well designed to fit to the physical world. Our approach provides means for representing the way of mathematical thinking.
We now show how mathematical concepts can be used to formalize the cognition process when going from a textual example to the mathematical model. Consider our coil example. The first step is to define the basic mathematical concept, i.e., concepts that are always valuable. For example the Pythagoras' theorem and computing the area of an rectangle.
The pre-condition to apply the Pythagoras' theorem is that the triangle has to have a rectangular angle. The post-condition is the equation that relates the sides of the triangle.

$$(pythagoras, \{rectangular\_triangle(a,b,c)\}, \{\}, \{equation(a^2+b^2=c^2)\})$$

The equation for the area of an rectangle is only valid for rectangles. Hence, the pre-condition has to ensure that the given values are for a rectangle.

$$(area\_rect, \{rect(a,b)\}, \{\}, \{area(rect(a,b)) = a \cdot b\}$$

Another example of a concept that is required for a large class of examples is how to build the sum of area values. This can be done by adding the areas and subtracting the overlapping value.

$$\left(\begin{array}{c} sum\_rectangles, \\ \{x, y, overlaps(x,y,o)\}, \\ \left\{ \begin{array}{c} (area(x) = A1) \in area\_rect(x)|| \\ (area(y) = A2) \in area\_rect(y)|| \\ (area(obj(x,y)) = O) \in overlap(overlaps(x,y,o)) \end{array} \right\}, \\ \{area(obj(x,y)) = A1 + A2 - O\} \end{array} \right)$$

The mathematical concept $overlap$ which is used in the previous concept of *sum_rectangles* can be defined as follows:

$$(overlap, \{overlaps(x,y,a)\}, \{\}, \{obj(x,y), area(obj(x,y)) = a\})$$

In the second step, we define the problem itself as a MCP:

$$\left(\begin{array}{c} \{rect(u,v), rectangular\_triangle(\frac{u}{2}, \frac{v}{2}, r), overlaps(rect(u,v), rect(u,v), u \cdot u)\}, \\ \{pythagoras, area\_rect, sum\_rectangles, overlap\} \\ \{\{equation((\frac{u}{2})^2 + (\frac{v}{2})^2 = r^2), area(obj(rect(u,v), rect(u,v))) = u \cdot v + u \cdot v - u \cdot u\}\} \end{array} \right)$$

This MCP obviously does not capture the third model resulting from formalization F_{III} of the coil example. For this purpose the MCP has to be re-written and additional concepts have to be introduced.
A MCP determines a possible sequence of the application of mathematical concepts that lead to one solution. A sequence is usually not unique. If we assume that no alternative actions are used, then the all sequences that explain the same goal state comprise the same set of mathematical concepts. Although, the mathematical concepts are used in a different order, they are identical with respect to the solution. This is not true if alternative actions are used. There might be a shorter sequence of mathematical concepts that lead to the same goal state. In this case the intended meaning of the cognitive process for modeling cannot be determined without further knowledge. Therefore, we restrict MCPs to well-formed MCPs which are defined as follows:

Definition 4.3 (Well-formed) *A MCP (I, M, G) is well-formed iff no two sequences $< m_1, \ldots, m_{i-1}, m_i, a_1, \ldots >$ and $< m_1, \ldots, m_{i-1}, m_i, b_1, \ldots >$ lead to the same goal state. Note that the mathematical concept m_i must have an alternative action.*

For well-formed MCPs it is sufficient to consider one mathematical concept sequence for each goal state in order to get enough knowledge about the (expected) cognitive modeling process. We use this result to show how such a sequence can be used to find a misunderstanding of the student during the modeling phase.
Consider again the coil example and its MCP. One concept sequence is:

$$\begin{gathered} < area\_rect(rect(u,v)), \\ area\_rect(rect(u,v)), \\ overlap(rect(u,v), rect(u,v)), \\ sum\_rectangles, \\ pythagoras > \end{gathered}$$

In this sequence concepts that call other concepts are written after the called concepts. This is due to the idea that a concept is understood if its "sub-concepts" (concepts that are called by the surounding concept) have been understood before.
From the first and second element of the sequence follows that $area(rect(u,v)) = u \cdot v$. The third element delivers $area(obj(rect(u,v), rect(u,v))) = u \cdot u$. By combining this result (which is done by the fourth element $sum\_rectangles$ we get $area(obj(rect(u,v), rect(u,v))) = u \cdot v + u \cdot v - u \cdot u$. The last element allows for deriving the result $equation((\frac{u}{2})^2 + (\frac{v}{2})^2 = r^2)$. Which perfectly corresponds to the pre-specified goal state.
Now assume that the student who has to do the modeling answers the the two equations $(\frac{u}{2})^2 + (\frac{v}{2})^2 = r^2$ and $A = u \cdot v \cdot 2$ (where A stands for the area of the cross shape). Whereas the first equation is correct, the second is not. The question now is, which principle has not been taken into account? The obvious answers is that the student has ignored the overlap of the two rectangles. We will now see how our approach is able to provide the student with the same answer.
For this purpose we first assume that if a concept is understood, then the output of the concept is specified by the post-conditions (taking the pre-conditions into account). If a concept is not understood, then the output is not specified. If a concept is comprised of sub-concepts, then its output maybe a sentence containing variables. For example, consider the concept $sum\_rectangles$ and assume that the $overlap$ concept is not understood. It follows that no expression for variable O can be determined. Therefore, the output value $area(obj(rect(u,v), rect(u,v))) = u \cdot v + u \cdot v - O$ is returned. But now the equation $A = u \cdot v \cdot 2$ does not contradict the output of the the MCP because they are equivalent if O is equal to 0. The system now can the answer that the concept $overlap$ was not understood and its output was assumed to be 0.
What we have used so far is the behavior of the concepts (which correspond to components in model-based diagnosis), the assumption that a not understood concept does not provide any information, and the assumption that two equations lead to a contradiction if they cannot be compared. Two equations can be compared, if there exists a function that maps one equation to the other.

5 Discussion and conclusion

The explanations ISAC generates so far, are completely related to the structure of mathematics, which is reflected by the mechanisms of ISAC, and by the knowledge, ISAC (and the student) are working on. This may be acceptable for parts of the specifying phase and for the solving phase; and it may be appropriate for the algorithmic parts in the narrow sense. But it is not satisfactory for the model phase, where the issue is to bridge the

gap from incomplete knowledge, unconcrete notions, unclear concepts represented in the students mind, to a formal model of the problem.
Therefore, we have introduced a framework that should allow for specifying the mathematical concepts and their interaction which are necessary to model and solve a given example. Having such a model enables us map wrong example solutions to cognitive parts that build-up the modeling and solving process. Hence, it should be possible to give the student a more focused feedback about the nature of the fault. I.e., the system can say that a concept, e.g., *overlapping areas of two geometrical objects have to be counted only once*, has not been used for solving the example (which was wrong).
The introduced framework should be understood as a basis for discussion. It has not be proven that all mathematical examples can really be expressed within this framework. Moreover, the semantics is currently only an informal one and is based on first order logic. Hence, future research has to (1) ensure that the framework can be used for as many examples as possible, (2) define a formal semantics, and (3) give complexity results. However, considering our mathematical coil example we could show that the framework delivers promising results.

References

[1] Bruno Buchberger. Algorithmic mathematics: Problem types, data types, algorithm types. Lecture notes, Research Insitute for Symbolic Computations, Johannes Kepler University, Linz, Austria, 1982. 320 pages.

[2] Johan de Kleer and Brian C. Williams. Diagnosing multiple faults. *Artificial Intelligence*, 32(1):97–130, 1987.

[3] Kees de Koning and Bert Bredeweg. Using GDE in Educational Systems. In *Proceedings of the European Conference on Artificial Intelligence (ECAI)*, pages 279–283, Brighton, UK, August 1998.

[4] Kees de Koning, Bert Bredeweg, Joost Breuker, and Bob Wielinga. Model-based reasoning about learner behavior. *Artificial Intelligence*, 117:173–229, 2000.

[5] Heinz-Christian Schalk et al. *Mathematik für Höhere Technische Lehranstalten*, volume 3. Reniets Verlag, Wien, Austria, 1998.

[6] Richard E. Fikes and Nils J. Nilsson. STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving. *Artificial Intelligence*, 2:189–208, 1971.

[7] Kenneth D. Forbus. Qualitative process theory. *Artificial Intelligence*, 24:85–168, 1984.

[8] Cliff Jones. A Rigorous Approach to Formal Methods. *IEEE Computer*, 29(4):20–21, April 1996.

[9] Simon L. Peyton Jones and David R. Lester. *Implementing Functional Languages, A Tutorial*. Prentice Hall International Series in Computer Science. Prentice Hall, New York, London, Toronto, Sydney, Tokyo, Singapore, 1992.

[10] Donald E. Knuth and P.B. Bendix. Simple word problems in universal algebra. In J. Leech, editor, *Computational Problems in Abstract Algebra*, pages 263–297. Pergamon Press, 1970.

[11] Walther A. Neuper. A 'calculemus-approach' to high-school math ? In Steve Linto and Roberto Sebastiani, editors, *Proceedings of the 9th Symposion on the Integration of Symbolic Computation and Mechanized Reasoning*, Siena, Italy, June 2001.

[12] Walther A. Neuper. *Reactive Uuser-Guidance by an Autonomous Engine Doing High-School Math*. PhD thesis, TU Graz, IICM – Software Technology, 2001.

[13] Raymond Reiter. A theory of diagnosis from first principles. *Artificial Intelligence*, 32(1):57–95, 1987.

Model-Based Reasoning for Tutorial Dialogue in Shipboard Damage Control[1]

Elizabeth Owen Bratt, Brady Clark, Zack Thomsen-Gray, Stanley Peters, Pucktada Treeratpituk, & Heather Pon-Barry

Center for the Study of Language Information
Stanford University
Stanford CA 94305-4115 USA
phone: 650-725-2319
{ebratt,bzack,peters,ztgray,pucktada, ponbarry}@csli.stanford.edu

Karl Schultz, David C. Wilkins, David Fried
Beckman Institute
University of Illinois
Urbana, Illinois 61801
{kaschult, dcw, fried}@uiuc.edu

ABSTRACT: The intelligent tutoring system based on the DC-TRAIN simulation requires specific information of various sorts about the student's performance during a session, in order to provide a focussed instructional review. The flexible dialogue architecture allows for dynamic system adaptation to student performance during the dialogue.

Keywords: simulation session action and knowledge representation for tutorial dialogue

1. Introduction

In this paper, we describe how our tutorial system extracts and uses information from the model of shipboard damage control provided by the DC-TRAIN simulation, and its summary of student actions compared to expert actions. Issues include designing appropriate representations and methods of accessing information from simulator sessions. We discuss our system architecture, and how our flexible dialogue management and intelligent tutor module provide effective instructional dialogue that adapts to each student's DC-TRAIN session and performance within the dialogue.

2. DC-TRAIN for Shipboard Damage Control

The DC-TRAIN system (Bulitko & Wilkins 1999) supports qualitative reasoning in training damage control assistants (DCA's) to manage shipboard crises appropriately, using general principles of how and where fires might spread, likely consequences of fires (smoke) and firefighting (floods), and tradeoffs in resource use involving strategically important compartments such as magazine rooms filled with explosive weapons and large vs. small compartments.

[1] This work is supported by the Department of the Navy under research grant N000140010660, a multi-disciplinary university research initiative on natural language interaction with intelligent tutoring systems

The DC-TRAIN system involves a simulation of a DDG-51 Arleigh Burke destroyer, its firemain, and the properties of its compartments with respect to fires. Student DCA's can use DC-TRAIN for running through scenarios to test their ability to handle the fire, flood, smoke, and firemain crises produced by various scenarios, in reasonable speed and with the most effective containment of the crises. A separate DCX (damage control expert) agent component can run at the same time as the DC-TRAIN simulation to provide the actions of an idealized DCA, based on rules for optimal response to each crisis. The DCX agent generates a comparison of the expert actions with the student's actions, in the form of a database table called the Expert Session Summary Graph (ESSG).

3. Tutorial System

As a further training component, we are developing a tutorial system to conduct after action review with the student DCA, based on the ESSG record of each of their sessions with DC-TRAIN. We discuss first some of the issues in extracting and representing knowledge about the student's DC-TRAIN session, then outline our overall system architecture, with more detailed discussion of our dialogue management and tutoring module's structure and strategies.

3.1 Knowledge Representation Issues

Ensuring that the ESSG contains adequate information for the tutorial component to review with the student has been an important design goal of recent work. Some of the necessary modifications to the ESSG have been:

- providing an explicit representation of the causal link between original crises in a scenario, and resulting crises caused by fires spreading, fires producing smoke, fire fighting efforts producing flooding, etc. This allows the tutorial system to present its discussion of related crises in a coherent fashion, and to have a measure of how severe the results were from a crisis from the scenario.

- for each action or report in the system, providing a representation of the main action or statement, plus values for each of the possible parameters. This allows the tutorial system to speak about comparisons of the student's actions with the correct action, as well as to have the means to aggregate related actions, either of the DCX or the student, for purposes of discussion. The aggregation may be necessary either for the system's plan of how to conduct the review, or for the system to respond to the student's questions. The discussion may be of what crises had similar mistakes, similar recommended actions, or similar use of resources such as the same repair teams being involved.

- providing a representation of the goals which a particular action may serve, from immediate and direct goals, to higher level ones.

- tolerance to unimportant differences from DCX, such as when the student chooses to start a different fire pump from the one that the DCX chose, but the student's pump choice meets all applicable constraints.

DC-TRAIN produces the initial representation of the ESSG information in a Microsoft Access database table, which we translate into a Java class for easy querying by our Tutoring Module (discussed in Section 3.4).

Beyond these issues of representing the ESSG information appropriately, we also are addressing issues of visualization of information for presentation during the tutorial session. A ShipDisplay lights up compartments with various colors indicating the type of crisis during DC-TRAIN. The tutorial system aims to use the same display to highlight items during discussion. This could be especially valuable for discussing items of spatial reasoning, such as the proper placement of fire boundaries and tracing the spread of a fire or flood and discussing considerations in managing that crisis at various stages of its development.

3.2 System Architecture

In this section, we discuss some models and techniques we used to deal with the complex structure of fire, flood, smoke, and firemain crises produced by various DC-TRAIN scenarios.

To facilitate the implementation of multi-modal, mixed-initiative interactions, we implemented our system within the Open Agent Architecture (OAA) (Martin et al. 1999). OAA is a framework for coordinating multiple asynchronous communicating processes. The core of OAA is a 'facilitator' which manages message passing between a number of encapsulated software agents that specialize in certain tasks (e.g., speech recognition).

Our system uses OAA to coordinate agents for the following components of the system:

- **Gemini** natural language understanding (Dowding et al. 1993). Gemini uses a single unification grammar both for *parsing* strings of words into logical forms (LFs) and for *generating* sentences from LF inputs. This agent enables us to give precise and reliable meaning representations which allow us to identify the discourse move types (e.g., a question) given a linguistic input or output; e.g., the question "What happened next?"' has the LF: (ask(wh([past,happen]))).

- **Nuance** speech recognition, which converts spoken utterances to strings of words. Nuance relies on a language model, which is compiled directly from the Gemini grammar, ensuring that every recognized utterance is assigned an LF.

- **Festival** speech synthesis, which 'speaks' word strings generated by Gemini.

- A Dialogue Manager, which coordinates inputs from the user, interprets the user's dialogue moves, updates the dialogue context, and delivers speech and graphical outputs to the user. This agent is discussed in detail in Section 3.3.

- A **Tutoring Module**, which identifies appropriate tutorial strategies (positive or negative feedback, follow-up questions, hints) in response to user input. This agent is discussed in detail in Section 3.4.

Agents 1-3 are 'off-the-shelf' dialogue system components (apart from the Gemini grammar, which must be modified for each application). We implemented agents 4 and 5 in Java specifically for this application. This OAA/Gemini/Nuance/Festival architecture has been deployed successfully in an unmanned helicopter interface (Lemon et al. 2001).

3.3 Dialogue Manager

3.3.1 Dialogue Management Architecture

The Dialogue Manager makes use of several recent ideas in dialogue modeling, described in detail in Lemon et al. (2001). It creates and updates an *Information State*, corresponding to a notion of dialogue context. Dialogue moves (e.g., an assertion) update the information state and are mixed-initiative. A dialogue move might send a response to the tutoring system, elicit feedback from the tutoring system, or prompt a follow-up question.

Figure 1 provides a sample of the type of tutorial interaction our system aims to support.

T=tutor, S=student	
T	There were two fires, and you put out both of them. You also handled one flood. The first fire was in the officer's baggage room. For a fire, what's the first thing you should do?
S	Send a repair team to set fire boundaries on the compartment.
T	No, you missed a step. What comes before setting fire boundaries?
S	I'm not sure.
T	Okay. First send a team to investigate. For the second fire, you also forgot to send a team to investigate. Okay. After setting fire boundaries, what should you do?
S	Order a repair team to fight the fire.
T	No, you missed a step. What should you do before sending a repair team to fight the fire?

S	Isolate the space.
T	Yes. Always electrically and mechanically isolate a space before fighting a fire. For the second fire, you also forgot to isolate the space.

Figure 1: Sample Dialogue

The Dialogue Manager includes two types of declarative knowledge: dialogue move types and the activity model. Dialogue move types are Java classes and include *assertion*, *wh-query*, *yes-no-query*, *yes-no-response*, etc.

The Activity Model is a hierarchical and temporal decomposition of tutorial activity. Unlike other aspects of the dialogue management architecture, only the Activity Model is specific to tutorial activity. The Activity Model includes *activity types*. Activity types are Java classes and include EXPAND_ONLY, SUMMARY (i.e, a statement that does not require a response), YN_QUERY (e.g., a question like "Ready to begin reviewing your session?"), and WH_QUERY (e.g., "What comes before setting fire boundaries?"). EXPAND_ONLY activities are more like placeholders for scripted sets of actions the tutor plans to perform. For example, the EXPAND_ONLY activity type 'start' corresponds to a sequence of actions: a YN_QUERY 'ready to begin', an EXPAND_ONLY node 'review session', and a SUMMARY ('goodbye'). Aside from EXPAND_ONLY, activity types represent primitive actions like asking a yes-no question.

The Dialogue Manager includes the following dynamically updated components:

- A *Dialogue Move Tree*: a structured history of dialogue moves and 'threads', plus a list of 'active nodes'.
- An *Activity Tree*: a temporal and hierarchical structure of activities initiated by the system or the user, plus their execution status.
- A *System Agenda*: the issues to be raised by the system
- *Salience Groups*: the objects referenced in the dialogue thus far, ordered by recency (Fry et al. 1998)
- *Pending List*: the questions asked but not yet answered
- *Modality Buffer*: stores gestures for later resolution

In the Dialogue Move Tree, each node is of a particular dialogue move type. Each node in the tree is a dialogue move, either by the tutor or by the student. Nodes in a dominance relation correspond to sub-dialogues; e.g., when the tutor asks a follow-up question in response to the student's answer to a question like *What should you do in response to a fire alarm?*

In the Activity Tree, each node is of a particular activity type. Nodes are constructed and added to the tree by the Tutoring Module (see Section 3.4 below). At the initial part of the session, the system creates a single EXPAND_ONLY node called 'start' and adds it to the tree. It then begins executing tasks. The activity 'start' is passed to a hashtable which returns the corresponding sequence of actions: a YN_QUERY 'ready to begin', an EXPAND_ONLY node 'review session', and a SUMMARY 'goodbye'. These nodes are all added to the tree and the YN_QUERY is asked. The tutor then waits for a response from the student. Based on that response, the tutor adds a new action to the tree and executes again or simply executes the next task.

3.3.2 Benefits

There are several benefits to our dialogue management architecture:

- The dialogue management architecture is reusable across domains. As mentioned, the same architecture has been successful implemented in an unmanned helicopter interface (Lemon et al. 2001). The activity model--- e.g., the properties of the relevant activities--- will have to be changed across domains.

- The Dialogue Tree/Activity Tree distinction allows one to capture the notion that dialogue works in service of the activity the participants are engaged in. That is, the structure of the dialogue, as reflected in the Dialogue Move Tree, is a by-product of other aspects of the dialogue management architecture; e.g., the Activity Model and the Tutoring Module. The Dialogue Tree/Activity Tree distinction is supported by recent theories of dialogue; e.g., Clark's (1996) joint activity theory of dialogue.

- The dialogue move types are domain-general, and thus reusable in other domains.

- The architecture supports multi-modality with the Modality Buffer. For example, we are able, in principle, to coordinate linguistic input and output (e.g., speech) with non-linguistic input and output (e.g., the user can indicate a point on a map with a mouse click or the system can illuminate a point on a map).

3.4 The Tutoring Module

3.4.1 Tutoring Module Components

The Tutoring Module processes the ESSG in order to discuss the student's DC-TRAIN session based upon the structure, causality, and behavior of its components (i.e., crises and actions). This information is used both to construct an overall tutoring strategy and to determine appropriate reactions to student input.

Before any tutor-student interaction takes place, the Tutoring Module devises the aforementioned tutoring strategy. This strategy composes the EXPAND_ONLY node 'review_session' which is added to the Activity Tree as a part of the 'start' node. This node defines which crises (out of the total set in the session) the tutor will discuss with the student. Crises are eliminated if the student performed perfectly in response to that crisis, or if a crisis with similar errors has been chosen for discussion. A given scenario may have many of the same type of crisis, so this preprocessing of the ESSG into a tutoring strategy seeks to avoid redundancy and keep the student interested.

The Tutoring Module must thus perform an analysis of the ESSG while constructing this strategy. Similar crisis types must be identified, as well as similar student errors. It is important to note that repeat errors are not only relevant in the strategy construction; they are also important to mention to the student during the tutoring dialogue itself (as seen in Figure 1). We hope to identify “exemplar” crises in the session. These are those crises that represent the poorest student performance on that given crisis type. These crises will make for the most interesting dialogues and provide the student with the most opportunity for learning. It may be the case that a student performs perfectly on all of a given crisis type save one and we may need to take this into account when evaluating the students performance.

As noted above, the Tutoring Module identifies appropriate strategies in response to user input. The Tutoring Module includes one type of declarative knowledge: a library of tutoring strategies. Figure 2 illustrates a tutoring strategy. For legibility, the key elements are presented in English rather than Java. Figure 3 formats the strategy in Figure 2 as a dialogue.

```
def_strategy discuss_error_of_omission_answer_incorrect

    : goal  (did_discuss_error_of_omission_answer_incorrect)

    : preconditions
        (i) the student's answer is incorrect
        (ii) the student's actions in response to the damage event included an error of omission

    : recipe
        (i) provide negative feedback to the student
        (ii) give the student a hint
        (iii) ask a follow-up question
        (iv) classify the student's response
        (v) provide feedback to the student
        (vi) tell the student the rule
        (vii) tell the student that the topic is changing
```

Figure 2: Sample tutorial strategy

T	The first fire was in the chemical warfare defense equipment storeroom No. 2. What is the first thing you should do in response to this crisis?

S	Send a team to isolate the compartment.
T	No, that incorrect. You missed a step. What should you do before isolating the compartment?
S	I don't know.
T	You should investigate the compartment. Let's move on.

Figure 3: Sample tutorial strategy dialogue

To initiate a tutoring strategy, the student invokes the Tutoring Module by responding to a question from the tutor; e.g., "What should you do in response to a fire alarm?" The system searches the library of a tutoring strategies to find all strategies whose preconditions are satisfied in the current context. Like the plan operators in other systems (e.g., Atlas/Andes; Freedman 2000), each tutorial strategy has a multi-step *recipe* (Wilkins 1988) composed of a sequence of actions. Actions in a recipe can be primitive actions like providing feedback or complex actions like embedded tutorial strategies.

The Tutoring Module utilizes information in the ESSG to decide which tutorial strategy is appropriate with respect to a student's response to a particular question. For example, in Figure 2, the Tutoring Module uses, in the preconditions on the application of the tutoring strategy, the information in the ESSG which classifies the relevant action as an error of omission, in addition to the classification of the student's response as an incorrect answer. The preconditions on other tutoring strategies will involve different combinations of action and response classification. Hence, it is the combination of the classification of a student's response (as correct, incorrect, etc.) and action in a DC-TRAIN session (as an error of omission, error of commission, etc.) which determine which tutoring strategy the Tutoring Module uses to teach the student.

The response classification mentioned above is not trivial. The student response to the tutor's query and the ideal student response as stored in the ESSG are in dissimilar forms. They are both translated into Command objects which represent the actions in question. The two objects can then be easily compared. Comparisons are made based on each relevant field of the object - the order type (e.g., setting flood boundaries), the agent (e.g., repair team 5), and the location (e.g., the laundry room). Student responses that do not match the ideal response in one field will be dealt with differently than student responses that do not match the ideal response in other fields. For example, a response with the wrong order type is a more serious error than a response with only the wrong agent.

The Tutoring Module makes uses of other aspects of the ESSG when tutoring the student; e.g., the causal structure of the model. Some actions in DC-TRAIN are steps in sequences of actions. For example, when fighting a fire the DCA must send a repair team to investigate the compartment, isolate the compartment, set fire boundaries, and fight the fire, in that order. If the student incorrectly omits one of these steps, while still succeeding in completing the other actions in the sequence, this is an error of omission. Students may also fail to identify the correct step in a causal sequence when responding to a tutor's question. The Tutoring Module uses this type of causal information contained in the ESSG to determine the appropriate type of hint (e.g., "You missed a step") to give to a student.

4. 0 Conclusion

Tutoring sessions use a complex representation of student DC-TRAIN performance, and structure it to provide an effective, focused review. The flexible dialogue architecture allows for the system to adapt to student performance during the dialogue, taking patterns of action from the DC-TRAIN session into account, both in formulating dialogue responses and in creating dialogue structure.

References

Bulitko, V.V. and D.C. Wilkins. 1999. Automated instructor assistant for ship damage control. *Proceedings of AAAI-99*

Clark, H.H. 1996. *Using Language*. Cambridge University Press.

Dowding, J., J. Gawron, D. Appelt, J. Bear, L. Cherny, R.C. Moore and D. Moran. 1993. Gemini: A natural language system for spoken-language understanding. *Proceedings of the ARPA Workshop on Human Language Technology*.

Freedman, Reva. 2000. Plan-Based Dialogue Management in a Physics Tutor.
Proceedings of the Sixth Applied Natural Language Processing Conference (ANLP '00).

Fry, J., H. Asoh and T. Matsui. 1998. Natural Dialogue with the Jijo-2 Office Robot. *Proceedings of IROS-98*: 1278-1283.

Lemon, O., A. Bracy, A. Gruenstein and S. Peters. 2001. *Proceedings Bi-Dialog, 5th Workshop on Formal Semantics and Pragmatics of Dialogue*:57-67.

Martin, D., A. Cheyer and D. Moran. 1999. The Open Agent Architecture: a framework for building distributed software systems. *Applied Artificial Intelligence* 13, 1-2.

Wilkins, D. 1988. *Practical Planning: Extending the Classical AI Planning Paradigm* . San Mateo, CA: Morgan Kaufmann.

Moving toward an Interactive Model Based Design Assistor (IMBDA)

Dr Mark Ratcliffe
Professor Chris Price

Department of Computer Science, UW Aberystwyth

This paper describes a proposed system to capture the design processes of novice students with an interactive, collaborative design tool to develop a case based diagnosis system capable of interpreting the students' work, modelling the design process and responding with useful advice.

Pedagogy, CBR, Design Processes

Overview

The Department of Computer Science at the University of Wales, Aberystwyth has been building a tool to produce a fully interactive, collaborative design capture and feedback system. Once complete it will be used to capture the design process of cooperating novice designers in order to refine our knowledge of the way in which the design learning process works. It will enable us to acquire a much better understanding of the student's perception of design and the learning process it involves.

The knowledge gained from working on a specified set of case studies will then be used to develop a case-based system capable of assisting novice engineers to develop higher quality designs.
The longer term aim is to develop a more generalised second-level system which has an underlying model of the design learning process and can thus break free of the captured cases and be capable of advising students in the general development of their designs.

The final system will be able to simulate other group players to give individuals experience of designing within a team, even if one is not present. It will be applicable to paradigms of software development other than that for which it was originally developed in particular to more general engineering projects that utilise computer aided design.

Background

Current teaching is less effective than it should be
Software development is a relatively young engineering discipline, yet there have been many fundamental changes in the techniques employed for the basic development of software systems. The latest paradigm, that of Object Oriented Design, has become increasingly popular and is now commonplace in most courses on software development. Object Oriented Development is claimed to be one of the most natural forms of development, modelling closely the way in which systems exist in the real world, yet there are still problems in teaching the technique. At a recent conference of the *Learning and Teaching Support Network for Information and Computer Sciences* [1], there was wide consensus that the success rate of this teaching is very poor. A similar international review of first year students' programming skills reached the same conclusion [2].

Many different approaches are used to assist novice programmers in their attempts to learn the design process. Unfortunately, as demonstrated by a recent paper [3], they all require experience in order to be truly successful:
"The first step in actual class design is to find the primary objects" [4]
"Identify the classes and objects at a given level of abstraction"[5]
"The content of an object model is a matter of judgement …" [6]
"As analysts experienced in [design…], we recognise certain patterns" [7]

and the list goes on. It is a chicken and egg problem. How are students supposed to apply judgement in the absence of experience?

Tutors' experience is a barrier to student learning
Whilst novice programmers might be able to recite the techniques necessary to approach the design process, there is no real substitute for experience, but therein lies a problem. Most instructors of design naturally base their tuition techniques on their understanding of the design process. This is based on years of valuable experience and it is the key element that the students lack. Unfortunately, although the tutor's experience is the key to their own success, it is also a complicating factor. The lack of experience on the students' part puts up a significant barrier between the tutor and student. It is often the very reason why the tutor cannot appreciate the real difficultly that the students face.
Cooperative Design is a good thing.

One helpful idea appears to be allowing students to cooperate. Work undertaken at the German National Research Centre for IT demonstrates the benefits of cooperative learning in design [8].

Cooperative design has the added advantage of causing students to justify their design decisions and reflect upon them. Research has shown that knowledge alone is not sufficient for successful problem solving in a domain: the student must also choose to use that knowledge, and to monitor the progress being made [9]. The learning and construction of new knowledge structures requires similar self-awareness and reflection.

The prototype system that we are developing at Aberystwyth supports cooperative design and keeps a log of students' design decisions. Whilst it is more likely that in a team of students, there will be someone who can make good design decisions, the lack of experience still applies.

What IMBDA will achieve
This project will help us better understand the learning processes involved in designing software. It will deliver software capable of enhancing the learning process for beginning software engineers and to do this will incorporate elements of both cooperative design and focused and relevant tuition.

The current developments being undertaken are timely, as many academics and industrialists are beginning to realise that they way we are educating software engineers is flawed. Ratcliffe coordinated a session on this in the USA early in 2002 [10].

At first, there seems to be little that can be done to substitute for real experience, other than one-to-one tuition that points out where a student is going wrong at the critical time (and determining the critical time is itself of course a tricky pedagogical issue). With the IMBDA system, however, we hope to provide the equivalent of this one-to-one tuition in the form of a software system. Thus we will maintain personalised advice while providing savings in terms of time and money.

Similar work to this project is already underway in the UK through the AESOP [11] project though this only records information; it is not applying the case based or model-based technology to solve students' problems. This is where IMBDA excels.

The Approach taken by IMBDA

The general approach in IMBDA is a multi phase development that can be broken down into four distinct stages.

Implementation of capture tool.
Wherever possible IMBDA has been designed to make use of existing software. It is expected to complete this phase of development within the next 12 months though it will be ongoing as it is extended to support the other phases of development.

Preparation of case studies and deployment of tool to work with them.
At the same time as developing the software, it is necessary to investigate suitable problem specifications to be given to at least 150 students to test and enhance their design skills. Once a worksheet has been assigned, IMBDA will be used to capture as much information as possible on how the students' design evolves. The case studies will be chosen to encourage team working to enhance communication and make it easier to capture the feedback loop fundamental to the learning process.

Capture of design learning process and development of case-based system
Each individual problem specification should produce a large set of captured designs. These case studies will be structured into a database to enable the tool to be developed into a case-based design assistor. This is a challenging task, involving representation of designs and of the decisions behind those designs. This must be combined with the ability to diagnose and repair the student's understanding of how to proceed with their design solution.

While a case-based system of such sophistication has not been attempted in the past, it should be achievable by use of case-based representation of designs, representing differences between designs as adaptations [12]. The monitoring, diagnosis and repair of student understanding using the case-base will require modelling of the student design process, and ground-breaking work in this area (although in a much simpler domain) has been carried out [13]. We will use that work as a starting point for developing appropriate modelling of students designing software.

New students will be able to work through the example sets using IMBDA as an intelligent advisor guiding them through the design process. Unlike a traditional student-tutor set up, IMBDA will be advising based on information gained from students working with similar experience to that of the user but with refinements gained through trial and error. It is hoped that the really significant factor is that the feedback will be at the same level as that of the student. It is expected that this will be a major step forward in the pedagogy.

Creation of model based system based on these experiences.
Once the case based system has been implemented, IMBDA will move into the final most adventurous phase, which is to take the design cases and factor out a more generalised model of the software design learning process thereby breaking out of the confines of the original case studies. This stage aims to develop and refine a generalised model of the learning process facing novice designers and identify effective techniques that can be used to overcome them.

Generalisation of the case-based IMBDA working on specific cases studies to an IMBDA capable of assisting novice programmers with a wider range of design problems is the most ambitious phase of the project. We expect to be able to abstract a set of general causal mechanisms appropriate to novice programmers from the case studies and from the experience with the case based system. We will then apply these to new case studies to generate for unseen problems the kind of support that the case-based system provides for the original case study problems. Stroulia does similar abstraction and reuse of design principles, albeit in a simpler domain [14].
Conclusion

We hope that the initial case-based system for software designers will prove a real asset in teaching software development by improving the quality of the educational experience. Most Computer Science Departments are only too aware of the low success in helping novice programmers develop their programming and design skills. This system will assist in speeding up the learning process for the learner by helping them gain knowledge that is usually only developed through extensive experience.

The model based system will prove a more general resource in the education and retraining of software developers ultimately enhancing the supply of good quality engineers to the software industry. This could prove particularly significant since the relative youth of software development and its enormous rate of growth inevitably means that over the next few decades there will be a significant amount of retraining required. Migrating to new development techniques is not easy. As the final system will be applicable to paradigms of software development other than that for which it was generally developed, the work undertaken through

IMBDA will assist in the retraining of these software engineers. The system will also be of use to other more general engineering projects that utilise computer aided design.

Bibliography

[1] 2nd Annual LTSN-ICS Conference, University of London, August 2001.

[2] M. McCracken et al., "A multi-national, multi-institutional study of assessment of programming skills of first-year CS students", report of an ITiCSE 2001 Workshop, *SIGCSE Bulletin*, December 2001.

[3] J.M. Maris & C. VanLangen, *A Design Tool for Novice Programmers*, Working Paper Series 00-01-April2000, http://www.cba.nau.edu/working_papers/papers&abstracts/_MarisVanLanLucy/Novice.htm

[4] D. Arnow & G. Weiss, *Introduction to Programming Using Java: An Object Oriented Approach*, Addison Wesley, Menlo Park, California, 2000, p.142.

[5] G. Booch, *Object Oriented Design with Applications*, Benjamin/Cummings, Colorado, 1991, p. 190.

[6] J. Rumbough et al., *Object Oriented Modelling and Design*, Prentice Hall, Englewood Cliffs, New Jersey, 1991, p47.

[7] P. Coad & E. Yourdon, *Object Oriented Analysis*, 2nd edition, Yourdon Press, Englewood Cliffs, New Jersey, 1991, p48.

[8] T. Holmer & I. Schummer, *A Tool for Co-operative Program Exploration*, ECOOP 2000 Workshop, Tools and Environments for Understanding Object-Oriented Concepts, June 12, 2000

[9] E.A Silver, "Foundations of cognitive theory and research for mathematics problem solving instruction" in A. H. Schoenfeld (Ed.), *Cognitive science and mathematics education* (pp. 33-61). Hillsdale, NJ: Lawrence Erlbaum, 1987

[10] M.B. Ratcliffe "Improving the Teaching of Introductory Programming by Assisting the Strugglers", The 33rd ACM Technical Symposium on Computer Science Education**,** Cincinnati, USA, February 27 - March 3, 2002.

[11] M. MacGregor, Pete Thomas and Mark Woodman, *AESOP (An Electronic Student Observatory Project)*: ITiCSE 2001, Innovation & Technology in Computer Science Education, Canterbury, Kent.

[12] T.R. Hinrichs & J. Kolodner, "The role of adaptation in case-based design" Proceedings AAAI-91, 1991

[13] K. de Koning, B. Bredeweg, J. Breuker and B. Wielinga, "Model-Based Reasoning about Learner Behaviour", *Artificial Intelligence*, Vol 117, No. 2, pp173-229, 2000.

[14] A. Goel and E. Stroulia, "Functional Device Models and Model-Based Diagnosis in Adaptive Design", Artificial Intelligence for Engineering Design, Analysis and Manufacturing, vol 10, pp355-370, 1996.

A case study of collaborative modelling: building qualitative models in ecology

Paulo Salles

Universidade de Brasilia, Instituto de Ciências Biológicas

Campus Darcy Ribeiro, Brasilia - DF, 70.910-900, Brasil

E-mail: paulo.bretas@uol.com.br

Bert Bredeweg

University of Amsterdam, Department of Social Science Informatics

Roetersstraat 15, 1018 WB Amsterdam, The Netherlands

E-mail: bert@swi.psy.uva.nl

Keywords

Collaborative Model Building, Qualitative Modelling/Reasoning, Distance Learning, Ecology

Abstract

Modelling is seen as a learning activity in itself and qualitative modelling environments start to play a role in this respect. However, building (qualitative) models is not an easy task. It is therefore necessary to develop support for teachers and students. This paper describes an experience in which Artificial Intelligence (AI) undergraduate students from the University of Amsterdam and graduate ecology students from the University of Brasilia were engaged in a collaborative model building activity. The objective was to build qualitative models about the carbon cycle and the greenhouse effect in GARP.

A questionnaire was used to obtain the students opinion about different aspects of the modelling effort. Almost all the students (94%) reported an increase in their understanding of the ecological problems after the modelling activity, (an observation that supports the idea of modelling as a learning activity in itself). In certain aspects, being an ecologist (and therefore possessing relevant domain knowledge) made some parts of the model building activity easier. For example, global identification of the processes involved. Contrary, the AI students found it easier to construct typical AI representations, such as subtype hierarchies. The most difficult task for both groups was to build a library of model fragments. Identifying quantities and their quantity spaces were also mentioned as difficult.

In order to improve the performance of the qualitative modelling environments the QR community has to put effort in developing authoring tools with explanatory facilities. The study reported here provides some insights on how to scaffold such model building tools.

1. Introduction

Model building is becoming an important educational activity. According to Forbus *et al.* (2001) it is important that students become modellers because during the modelling process they have to articulate relationships between entities and dependencies between their beliefs. This is important for both understanding the phenomenon being modelled and in developing a broader understand of complex, interrelated systems. This way, models provide means to externalise thoughts and to support questioning, discussion and justification of decisions. Finally, modelling provides students with practice in using formal representations, a skill needed for mastering mathematics and programming.

Qualitative Reasoning (QR) has a role in introducing modelling into the classroom. Historically QR systems are linked to education (e.g. SOPHIE, Brown *et al.*, 1982, and STEAMER, Hollan *et al.*, 1984). Recently, a new generation of QR related tools is being developed. During the last international workshop on QR (San Antonio, US, 2001) a number of papers illustrated that new approach: *Mobum* (Bessa & Bredeweg, 2001) and *VisiGarp* (Bowers & Bredeweg, 2001), *Vmodel* (Forbus *et al*., 2001), and *Betty's Brain* (Leelawong *et al*., 2001).

As QR-related learning environments become available, and more people start building qualitative models, it becomes important to actively develop tools to support such model building activities. To carry out an exploratory study on this topic, a collaborative modelling effort was conducted with students from the University of Amsterdam (UvA) and the University of Brasilia (UnB). These students designed qualitative models about the carbon cycle and the greenhouse effect, using the qualitative reasoning engine GARP (Bredeweg, 1992). They worked on a pencil and paper basis, and at the end of the course some of the students actually implemented their models in GARP. In order to investigate difficulties students found during the collaborative modelling effort, a questionnaire was completed by the students about different aspects of the modelling process. Their answers give us indications of how to further develop the use of qualitative modelling in the classroom.

This paper first presents the experimental context of the collaborative modelling activity. Second, it briefly discusses the domain knowledge the students had to work with. Third, the activities are described that were followed in order to have the students construct qualitative models and simulations. Fourth, the results obtained from the answers given by the students to the questionnaire are discussed. Finally, the lessons learned from of this experience are discussed.

2. The Experimental Context

The exploratory study described here involved 10 undergraduate and MSc students at the University of Amsterdam (UvA) enrolled in the discipline 'Model Based Reasoning' (MBR) and six MSc and PhD students at the University of Brasilia (UnB) enrolled in the discipline 'Models in Ecology'. Due to their different backgrounds, the approach each group took to qualitative modelling was somewhat different. UvA students took it in terms of an artificial intelligence curriculum and UnB students took the modelling effort in the context of an Ecology curriculum. UvA students had a good introduction to the QR literature in their MBR course, whereas UnB students had only basic knowledge on that. The lectures also prepared a tutorial on GARP, particularly focussing on the notion of model fragments and qualitative behaviour graphs.

Five groups were formed, each consisting of two Dutch and one Brazilian student[1]. The overall modelling problem was divided into sub-problems and each group had to tackle a specific sub-problem. In order to facilitate the interaction and communication the e-group facility offered by Yahoo! was used (mbr-ecology). All students and lecturers were subscribed to the e-group. Thus, participants could communicate with each other using regular email as well as other Computer Supported Communication (CSC) tools provided by the Yahoo! e-group facility.

In order to evaluate the modelling activity a questionnaire was used consisting of 41 questions, including personal characterisation, course evaluation, the modelling effort, and the collaboration.

[1] One group had 2 Brazilian students.

3. The Domain

An important area in ecology is nutrient cycling. Among them, the carbon cycle is particularly relevant because it includes a very broad set of phenomena and involves the interaction of biological, physical and chemical processes related to the production and use of organic matter. One of the most interesting aspects of this cycle and a big issue nowadays is the fact that compounds of carbon, specially the carbon dioxide (CO_2), retains heat and therefore affects the climate – the greenhouse effect. One of the tasks for the students was to acquire the knowledge relevant to this domain (see also next section). Figure 1 is a picture found by the students on the World Wide Web that illustrates the problem situation[2].

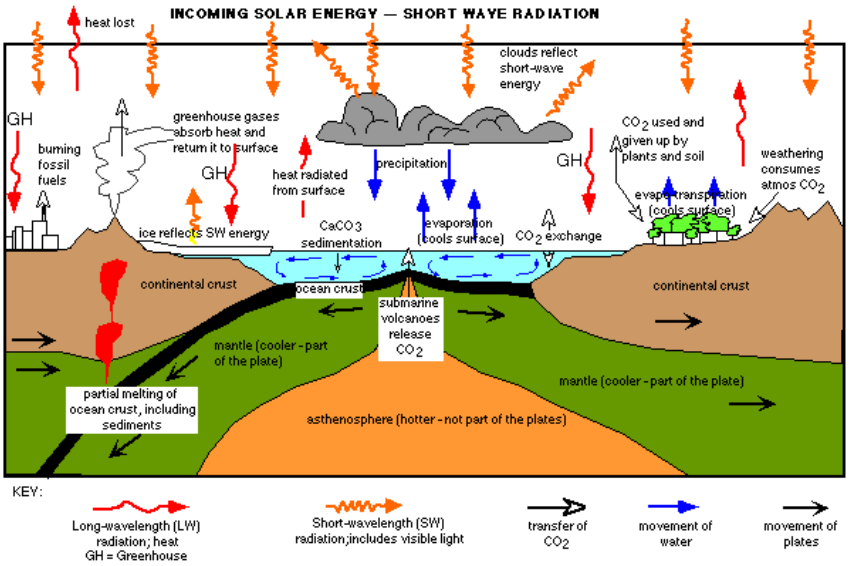


Figure 1: one of the many pictures found by the students illustrating the problem situation

4. The Model Building Method

The procedure described below was used to have the students conduct the required model building activities. Each activity was supposed to take one week, and students were expected to spend approximately 20 hours on the course during a week (including participating in seminars). For each activity, the teams had to produce written documents discussing their results and share this with the other students by placing the documents in the (mbr-ecology) e-group.

3.1 Starting the Collaboration.

Here the idea was that students should get to know each other, particularly to 'meet' the persons from the other university. Each student therefore had to perform the following three tasks:

- Register at Yahoo! and join the e-group (mbr-ecology) that was made by the lectures for this purpose.
- Send an introductory message to all the e-group members, particular stating: who you are, what you are studying, why you are interested in the modelling course and what your expectations in this respect are.
- Submit at least two bookmarks to the (mbr-ecology) e-group concerning 'global heating' (and/or the 'greenhouse effect'). The idea was that the pages referred to by these bookmarks would form the group's initial overall knowledge-based (understanding) of the problem.

In addition, and mainly to provide focus, the lectures gave the students a technical paper discussing some of the most important aspects of the 'carbon cycle' (Grace, 2001).

3.2 Form Teams and Assign Subsystems

Based on the information provided by the students, teams were formed to tackle the sub-problems that constitute the overall problem (the greenhouse effect). Notice that this step already enforced students to decompose the main problem into a set of sub-problems even though their knowledge on the domain was limited at this point. In order to prevent a potential deadlock the lecturers deliberately intervened both

[2] http://www.acad.carleton.edu/curricular/GEOL/DaveSTELLA/climate/climate_modeling_1.htm

concerning the problem decomposition and the forming of teams. After all, this step was an important one and needed to be solved in order to progress with the main model building activities.

Five themes were identified and each group started working on one of them: (a) a global model about the carbon cycle, (b) forests, (c) water and oceans, (d) human activities, like industries, transport, agriculture, and (e) the greenhouse effect.

3.3 Process Domain Related Publications

In order to learn more about their specific part of the overall problem teams had to study the domain related material (WWW pages, including some online articles) and produce a four page written document explaining and discussion their part. As all documents, this document had to be submitted to the (mbr-ecology) e-group at Yahoo! so that all teams could read about the knowledge acquired by the group as a whole[3]. Notice that each team consisted of students from both universities. During the seminars (locally at each University) group members had to present and discuss their ideas with the members from the other teams. This had two goals, first to share insights among teams and second to reduce too much diversity between the groups (part of the discussion focussed on the relations between the sub-problems).

Students were also instructed to download the qualitative simulation software (GARP and VISIGARP) and make sure that the software worked properly on their computers[4].

3.4 Structural Model and Global Behaviours

To arrive at a qualitative model of the systems under study, the model building activity should continue with a relatively strong focus on the knowledge representation underlying the qualitative simulator that we intended to use. The idea was to divide that goal into four steps (see also below). The first step consisted of three sub-activities:

- Structural model (objects and relations, e.g. part-of and is-a). Basically, a concept map including all the entities relevant to the problem organised in a subtype hierarchy. In addition definitions of structural relations, such as part-of, contains, etc, between those entities as far as needed (an example constructed by students is shown in Figure 2).
- Global description of behaviour (processes). Textual oriented descriptions of typical behaviours, in fact processes (e.g. respiration) or agents with certain behaviour (e.g. farming).
- Scenario's and related behaviour graphs. Develop two scenario's using the previously defined objects and behaviour descriptions and show how those lead to a particular behaviour graph relevant for understanding the issues concerning the problem of global heating.

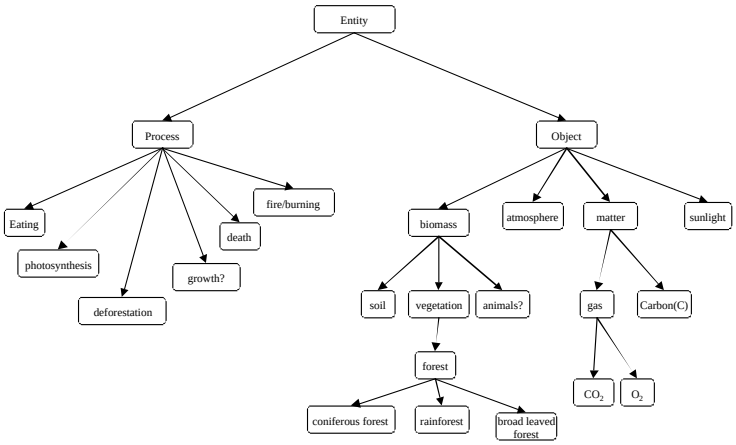


Figure 2: an example of an initial concept hierarchy constructed by students

[3] There were some technical problems with uploading and downloading MSWORD files. This caused some delay. However, soon students discovered that PDF and RTF formats could be handled properly by Yahoo!. Consequently DOC files were not used anymore.

[4] At the time of the course we did not yet have easy to use model building software, such as HOMER (Bessa Machado & Bredeweg, 2002). MOBUM (Bessa Machado & Bredeweg, 2001) was implemented as a demo, but was not stable enough to support a broad model building effort. Thus, only students with knowledge of PROLOG could be expected to actually build simulation models in GARP. Notice that currently this situation has changed. E.g. HOMER is a fully implemented workbench and can be used to build qualitative models. For details see WWW pages: http://web.swi.psy.uva.nl/projects/GARP/

3.5 Detailed behaviour model

The second step towards a specific qualitative model is to further detail the behavioural aspects. This consisted of three sub-activities:

- Define quantities.
- Define quantity spaces for each quantity and point out important landmarks.
- Construct an influence diagram (a 'causal model'). Using the previously defined quantities specify how they are causally related, mainly using the notions of influences and proportionalities (see, Forbus, 1984) (an example is shown in Figure 3).

Students also had to further detail the 'global behaviours' defined during the previous step. The idea being that this should lead to a first global description (using text) of the model fragments that will be part of the final model.

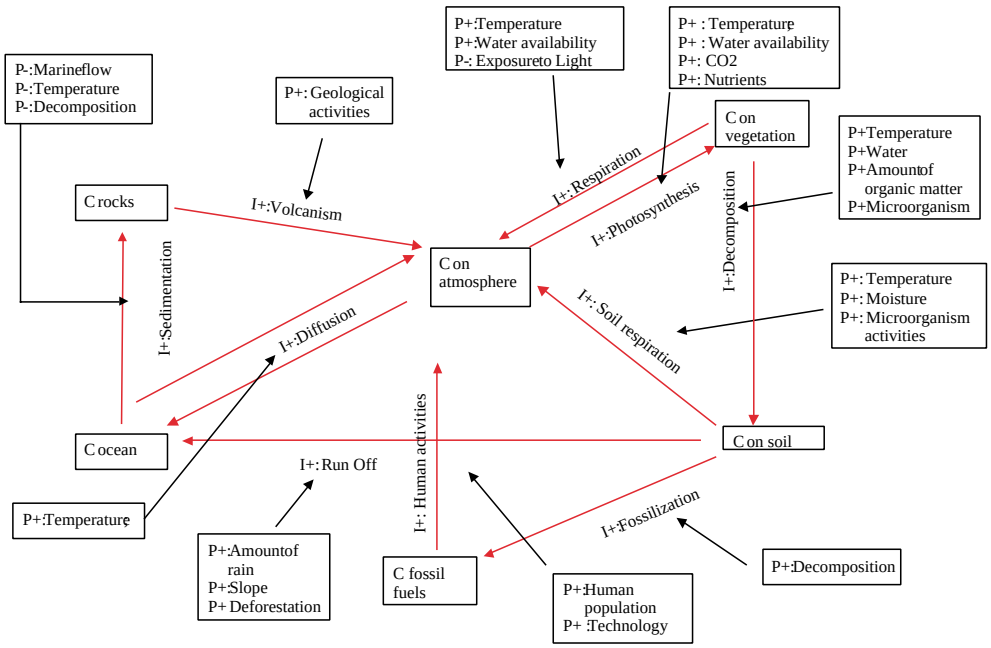


Figure 3: an example of an influence diagram constructed by students (stressing the fluxes)

3.6 Detailed specification of model fragments

The third step consisted of constructing detailed descriptions of the required model fragments. Although this step was performed using paper and pencil the students had to follow a specific syntax (provided by the lectures). Students who were more experienced with PROLOG were encouraged to formulate their model fragments directly in GARP, however, without running the simulator. The focus of this step was to conceptually clarify the set of model fragments (and not to focus on the overall effect of those model fragments on the behaviour graph potentially generated by the simulator).

3.7 Towards Detailed Implementation and Running a Simulation

During the fourth step the idea was to actually run models using the simulator. This mainly included the following sub-activities:

- Defining and implementing the possible scenarios (input systems).
- Analysing behaviour graphs generated by the simulator.
- Debugging and finalising the library of model fragments.

3.8 Writing Documentation

Finally students had to write a report discussing their model. The goal was not to simple copy and past the documents produced before, but to examine them and reformulate these documents following all the discoveries and modifications made during the model building activities. In other words, a description of the latest results had to be produced by each team.

5. The Modelling Effort

"*A change of reasoning and thinking*"

In the questionnaire there was a set of questions related to building the model. Some of these questions explored each of the main points in the models, like identifying objects, quantity spaces and so on. We asked the students to comment on how difficult they found it to develop each part of the models. The results are presented below.

Degree of difficulty – When the students were asked to define how difficult it was to build qualitative models, the answers varied. All UvA students mentioned an intermediate ('medium') level of difficulty. They made comments like qualitative modelling represented for them a "*change of reasoning and thinking*", and that once they understood the process, they found it doable. One UvA student said "*it took me a while to make the distinction between 'modelling' and 'programming'."* Among UnB students, one third found it 'more or less easy'. The tutorial prepared by the lectures made it easy for them to represent the basic processes. Also, the implementation part of the modelling effort was not their main concern.

Understanding the problem – Understanding the general problem (carbon cycle and the greenhouse effect) in the beginning of the modelling activity was 'more or less difficult' for one third of UvA students and more than two thirds of UnB students. One of the students said, "*the start was difficult. After that when you finally figured out what is needed, it was easy*". Some students referred to previous studies in biology, physics, earth sciences and chemistry at secondary school, which confirms the expectations that qualitative modelling draws on common sense and incomplete knowledge. Another one said understanding the problem was "*difficult, the theory didn't help at all, you just needed to look at what GARP needed*."

Identifying the most relevant aspects – For 40% of UvA students and 84% of UnB students, it was 'more or less difficult' and 'difficult' to identify the most relevant aspects of the problem. UvA students referred to a consult via email to their Brazilian partners "*to have a confirmation that we had chosen the right subjects*".

Identifying typical situations – Nobody found it 'easy' or 'more or less easy' to identify typical situations and to define initial scenarios. However, it was a bit more difficult for UvA students than for UnB students – maybe because of their background. One said it became "*easier as a result of identifying the most relevant aspects of the problem*".

Describing the system's behaviour – Asked about imagining the overall behaviour of the system and drawing a state-graph, the two groups gave different answers. For 84% of UnB students it was considered 'medium', 'more or less easy' and 'easy'. For 60% of UvA students it was 'more or less difficult' and 'difficult'. A UvA student said "*difficult mostly because of lack of ecological / chemical fore knowledge*" Another one mentioned the fact that the whole problem was "*fragmented over different groups*". This part of the modelling problem seems to be domain knowledge related and the difficulties are bigger in the beginning of the modelling activity.

Identifying physical objects – None of the students found it 'easy' or 'difficult' to identify the physical objects involved in the problem. For 84% of UnB students and 80% of UvA students the task was 'more or less easy' and 'medium'. One of the AI students of UvA said "*The difference between objects in GARP / Prolog and a OOP language as Java make some conceptions hard.*"

Representing objects and model fragments in isa hierarchies – In GARP, objects and model fragments are organised in isa-hierarchies. We asked the students to evaluate difficulties in building up such hierarchies. Similar distribution of answers was observed in the two groups. Half of both UvA and UnB students groups considered it 'more or less easy' and 'easy'; the other half considered this task 'medium', 'more or less difficult' and 'difficult'. Some UvA students mentioned their previous experience in doing this type of knowledge representation, whereas these were new concepts for the UnB students.

Identifying causal relations – A fundamental part of the qualitative modelling process is to identify causal relations and to draw diagrams of influences. We asked the students to include direct and indirect influences in their causal models. Opposite perceptions came up from this question: 90% of UvA students

said the difficulty level was 'medium', 'more or less difficult' and 'difficult', whereas for all UnB students it was 'medium', 'more or less easy' and 'easy'. We believe that the ability for explicating causal relations is very well related to domain expertise. Some interesting remarks made by UvA students refer to their difficulties: "*especially problematic was the distinction between indirect / direct influences*" indicates that they found it difficult to understand / identify processes. Another UvA student mentioned time scale problems, which are interesting and difficult aspects of ecological modelling: "*difficult, so difficult if this are going on all the time or at the same time*". Implementation is also an issue, as pointed out by a UvA student: "*this is the point were you need the domain knowledge and a lot of modelling knowledge*".

Identifying processes – Identifying processes is crucial for building qualitative models in GARP. Asked about how difficult it was, 80% of UvA students answered 'medium', 'more or less difficult' and 'difficult', whereas 100% of UnB students answered 'medium', 'more or less easy' and 'easy'. One of the UvA students said it was "*difficult, because many processes also occur in other subsystems*". Similarly to the previous question, domain knowledge is important for the students to identify processes.

Identifying quantities – None of the students found it easy to identify quantities and to define quantity spaces. For two thirds of UvA students and for half of UnB students, this task was 'medium', 'more or less difficult' and 'difficult'. Their comments are helpful for understanding their difficulties: "*more or less difficult, there are too many quantities, difficult to choose the relevant ones*". It is "*difficult to identify quantity spaces, it is hard to imagine. Would there be a maximum or not? Is zero an option?*" Another student said "*Difficult, it is hard to say when something is normal, high or maximum. Most of the times you could only say it's positive, normal or zero.*"

Creating model fragments – Knowledge about objects, quantities, relations, conditions for things to start and to stop and causal relations is represented in model fragments. They are the fundamental unity of the library that encode knowledge in GARP models. Therefore, creating model fragments is probably the most important part of the modelling activity. It is also one of the most difficult parts: 90% of the UvA students and 83% of the UnB students said building model fragments was 'medium', 'more or less difficult' and 'difficult'. It is "*difficult to translate the causal relations into a model fragment, because this has certain limitations on the representation you can use*" said one of the students. "*There you need to have a great understanding of GARP*", said another one. However, "*it helps when you get an example of the subject you are modelling*" suggested a third student.

The most difficult part – Creating model fragments was considered the most difficult part by 60% of UvA students and 50% of UnB students. 'Understanding the general problem', in the beginning of the modelling effort, came next: 30% of UvA students and 17% of UnB students selected this option. Some comments are worth to mention. The most difficult part was "*getting a good overview of the domain, especially the perspectives one can have of the systems*". "*Thinking about the relevance of things*" was also mentioned. A UnB student said, "*it was difficult in the beginning to understand the problems and the objectives of the modelling*". Identifying 'physical objects', 'typical situations and scenarios', and 'building causal models' were not mentioned by any student as the most difficult part.

The easiest part – Among the ecologists, the activity that received more votes (34%) was identifying the processes. A student said, "*when you know what needs to be done, it works very fast. Understanding GARP makes it easy*". For 20% of UvA students and 17% of UnB students constructing isa-hierarchies was the easiest part of the model building process.

6. Is it Worth to Build Models?

We asked the students to evaluate their knowledge about the carbon cycle and greenhouse effect and QR, having read the literature, BEFORE starting the model building and AFTER finishing the model. Building qualitative models had a positive effect on the learning process about the ecological problems for all the Dutch and for two thirds of the Brazilian students, according to their answers to the questionnaire.

The most impressive results here are: UvA = 40% beginners before, and 0% after; 0% was 'more or less expert' before, and 40% said so after. UnB = 50% was 'more or less beginner' before and 0% after; 34% was 'more or less expert' and 50% was 'more or less expert' and 17% 'expert' after. Some UnB students

(34%), who were not really involved in the modelling activity, said their knowledge on the domain did not improve.

We also presented the students a statement saying "building a qualitative model made me understand better the problem", and 60% of the UvA students and 83% of the UnB students said they 'agree' and 'fully agree'. One of the UvA students said "*fully agree, it gave me insight in the causality and the different perspectives from which we can look at the problem*". A different view was presented by another student, who said it "*made me able to **abstract** the problem*".

7. About the Collaboration

An UvA student said "*I think it is relevant to collaborate with people who have a different background (biology) but the fact that they are far from each other and in different countries wasn't relevant at all*" and others went on the same line. The UnB students were more excited about the international collaboration, something they are not used to. Asked about how effective was the collaboration the two groups had different opinions. The UvA students found it was not effective, whereas the UnB students found it effective.

Why is it that ecologists found the interaction 'effective' while their partners did not? An overall evaluation of the collaborative modelling effort shows that the students had an uneven experience. Five groups were formed, including students from both universities. One group had a strong interaction, and eventually they implemented part of their models in GARP; three groups had some interaction, but this was not regular; and one group had no interaction between the students. There are some elements that explain that. First, UvA students started the modelling effort earlier than their colleagues in Brazil. Therefore, they were not at the same stage of the modelling process and for some groups it was difficult to catch up. Given that e-mails exchanged in the e-group go to all the members, even for those that did not interact within their groups there was some sort of feedback. Second, there were problems with the language. Brazilian students have more difficulties with written English, and messages did not flow smoothly and quickly. Third, those groups that were not interacting could finish their work separately despite of their different background. AI students built their models using their basic knowledge of the ecological problem. Ecology students had some help (from the lecturer and from a tutorial on modelling) in order to design their models, but most of them did not try to actually implement their ideas in GARP. Finally, this was our first tentative in doing such interaction between students via internet, and we could not anticipate all the problems, prepare all the required didactic material beforehand, and adjust the timetable of the modelling activities for both courses.

Even though, 75% of all the students expressed that we should continue with the international collaboration in the future. As one UvA student said, "*yes, it was nice and it has never been done before*". The students suggested also to make a longer interaction: "*the contact should be made soon, the response should improve*" and "*increase the duration (time) of collaboration*".

8. Discussion

This exploratory study confirms the idea that modelling is a valuable learning activity, and suggests that collaborative modelling involving students with different background may add some extra value to that educational activity. We describe the collaboration between Artificial Intelligence (AI) students from UvA and Ecology students from UnB in order to build qualitative models of the carbon cycle and the greenhouse effect, using the representational schema adopted in GARP.

After the modelling effort, these students answered questions about general aspects of themselves and the course, and specific questions about the qualitative model building process. All in all, modelling is a doable activity for them, after understanding what they had to do. A student said qualitative modelling represented a "*change of reasoning and thinking*", an interesting comment for our reflection.

For the majority of the students understanding the problem they had to model and identifying the most relevant aspects were more or less difficult activities. Knowledge of the domain helps, but still this can be more or less difficult. Specific knowledge was important also for identifying typical situations the system may go through, and for describing system behaviours. Our study shows that AI and ecology students had

opposite opinions about how difficult it is to describe behaviour. The same divergence of opinions we found asking them about identifying objects and organising them in isa-hierarchies. AI students are used to that and find it an easy task. Ecology students were not used and had some difficulties to create such representations.

Probably the task that requires more domain knowledge is to build the causal models that underlay qualitative models. Once again, the two groups of students presented different opinions about this task. Almost all AI students marked 'medium' to 'difficult', and almost all Ecology students marked 'medium' to 'easy'. Some problems were mentioned by them, such as the fact that the distinction between direct and indirect influences is not always clear, processes are not easy to identify, and some aspects particularly related to ecological systems. Among them, processes that happen all the time and others that happen in particular moments; processes that occur in all the parts of the system and other that occur only in some parts.

The students mentioned also difficulties for identifying quantities and their respective quantity space. One student said in order to identify quantities and quantity spaces, "*even more perception and interpretation is needed*". In fact, this is a big issue in ecological modelling. Notions like 'boiling temperature', full of meaning in domains such as physics are not easily found in ecology. The solution our students found was to assume simple quantity spaces like {zero, plus} or {minus, zero, plus} for most of the variables.

Building model fragments was considered the most difficult activity of the modelling effort. Given that these partial models encode in a specific language representations of objects, quantities, conditions, relations, situations and processes, that is, the core of the knowledge being modelled, one can understand why the students answer that way in the questionnaire.

The students recognise that it is worth to build models for their learning process. After investigating the individual progress of the students reported in each questionnaire, we noted that 100% of students from UvA reported an increase in their knowledge about the ecological problems studied. Among the UnB students, 67% reported an increase, whereas 34% said their knowledge about the problem did not increase. In the overall evaluation, these latter students were those who made less effort in the course. So we can say the majority increased their understanding of the problem.

Students engaged in collaborative modelling sometimes act as teachers and sometimes as learners. As a UvA student said, "*maybe it is a good idea if the ecologists put more emphasis on explaining the modellers their domain knowledge. Then the modellers could spend more time on making the models and hierarchies, and maybe explain things about that to the ecologists*", the essence of collaborative modelling.

This study was not intended to be prescriptive, but we can organise some suggestions we captured and learned from our students. First, it is important to provide support for collaborative modelling. In the beginning, keep the focus on the expert, imagining the overall behaviour, identifying processes and drawing causal models. Describing behaviour in (qualitatively relevant and different) states, identifying quantity and quantity spaces, that is, understanding the problem is crucial for the success of the modelling effort.

When planning the collaboration, it would be a wrong decision to separate understanding the domain knowledge from 'how to implement' that knowledge. There is a pragmatic aspect mentioned in two different occasions to be explored. One student said modelling is "*difficult, the theory didn't help at all, you just needed to look at what GARP needed*" and another one said, "*when you know what needs to be done, it works very fast. Understanding GARP makes it easy*". These results suggest that the knowledge representation should come along with thinking about the ecological system and what is relevant to model.

As expected, the strength of the interaction varied among the groups involved. One group had a strong interaction and eventually implemented running models. One group did not interact much and did not produce running models. In between, three groups existed with a reasonable amount of interaction. The perception of the Dutch and Brazilian students about the effectiveness of this experience was nearly the opposite: for the former, it was not effective, and for the latter it was effective. We explain this paradoxical as a result of different schedules, backgrounds and proposals for the two groups.

Concluding, it is worth to continue with collaboratively building qualitative models. This activity provides the students with a better understanding of the problems. Skills acquired during the modelling effort will be useful in their professional future. All students expect for one ecologist, found that qualitative models are 'very useful'. They say why: "*It is convenient as a predictive tool in management strategies and decision making*". Another ecologist said qualitative models are "*very useful for formulating the reasoning about a problem*". It is also important to get people from different countries together. In our case, we intend to give more time for the students to get know each other better and to exchange cultural experiences. We believe that cross-fertilisation of ideas may improve learning opportunities.

Acknowledgements
We would like to thank the UvA and the UnB students for their collaboration and their willingness to shared their opinions and suggestions with us.

References

Bessa Machado, V & Bredeweg, B. (2001) Towards Interactive Tools for Constructing Articulate Simulations. Proceedings of the International workshop on Qualitative Reasoning, QR'01, pages 98-104, San Antonio, Texas, USA, May 17-19. Gautam Biswas (editor).

Bessa Machado, V & Bredeweg, B. (2002) Investigating the Model Building Process with HOMER. Proceedings of the International workshop on MBS/QR at ITS2002 (this volume).

Bouwer, A. & Bredeweg, B. (2001) VisiGarp: Graphical Representation of Qualitative Simulation Models. In J.D. Moore, G. Luckhardt Redfield, and J.L. Johnson (eds.), *Artificial Intelligence in Education: AI-ED in the Wired and Wireless Future*, pp. 294-305, IOS-Press/Ohmsha, Osaka, Japan.

Bredeweg, B. (1992) *Expertise in Qualitative Prediction of Behaviour*. PhD thesis. University of Amsterdam, Amsterdam, The Netherlands.

Brown J.S.; Burton, R. & de Kleer, J. (1982) Pedagogical, Natural Language, and Knowledge Engineering Techniques in SOPHIE I, II, and III. In Sleeman, D. & Brown, J.S. (eds.) Intelligent Tutoring Systems. London, Academic Press.

Forbus K. (1984) Qualitative process theory, Artificial Intelligence, Vol 24, Issue 1-3, pages 85-168.

Forbus, K.; Carney, K.; Harris, R. & Sherin, B.L. (2001) A qualitative modeling environment for middle-school students: A progress report. In Biswas, G. (ed.) Proceedings of the Fifteenth International Workshop on Qualitative Reasoning, St. Mary's University, San Antonio, TX.

Grace, J. (2001) The Garbon Cycle. http://www.ierm.ed.ac.uk/ierm/teaching/ccycle/ University of Edinburgh.

Hollan, J. D.; Hutchins, E. L.; & Weitzman, L. (1984) STEAMER: an Interactive Inspectable Simulation-based Training System. *AI Magazine*, vol. 5, no. 2, p. 15-27.

Leelawong, K.; Wang, Y.,; Biswas, G.; Vye, N. & Bransford, J. (2001) Qualitative Reasoning Techniques to support learning by teaching: The Teachable Agents Project. In Biswas, G. (ed.) Proceedings of the Fifteenth International Workshop on Qualitative Reasoning, St. Mary's University, San Antonio, TX.

Salles, P. & Bredeweg, B. (1997) Building Qualitative Models in Ecology. In Ironi, L. (ed.) *Proceedings of the 11th. International Workshop on Qualitative Reasoning (QR'97)*. Instituto di Analisi Numerica C.N.R., Pubblicazioni no. 1036 , Pavia, Italy.

Salles, P. & Bredeweg, B. (2001) Constructing Progressive Learning Routes through Qualitative Simulation Models in Ecology. In Biswas, G. (ed.) *Proceedings of the 15th. International Workshop on Qualitative Reasoning (QR'01)*. Saint Mary's University, San Antonio, TX, USA.

Walker, D.H & Sinclair, F.L. (1995) A Knowledge-based Systems Approach to Agroforestry Research and Extension. *AI Applications*, 9(3): 61-72.

Learning with qualitative models and cognitive support tools: the learners' experiences.

Julie-Ann Sime
Centre for Studies in Advanced Learning Technology,
Educational Research Department,
Lancaster University,
Lancaster, LA1 4YL, U.K.

J.Sime@lancaster.ac.uk
Tel: +44 1524 594726
Fax: +44 1524 592914

Abstract
This paper looks at the learners' experiences when learning with CPRODS, an interactive learning environment based on 6 qualitative and quantitative models. The learners spent 2 hours during the study using the learning environment, completing the evaluation questionnaire and tests of knowledge. Analysis of the data has revealed some interesting comments e.g. on the interface design, functionality, learning styles. Recommendations for the future have been derived from this study. The learning environment has also been evaluated to see if it has all the features that Kolb (1984) says are necessary for experiential learning, or learning by doing. The five learner support tools do provide most of the learner support that Kolb prescribes, although reflection on the learning process is identified as a weakness of the learning environment. On the other hand the support provided for the learner is a strength. The qualitative models are readily accepted by learners and seen as worthwhile although more could be done to explain the reason for using 'rough calculations' in problem solving. There are also some design recommendations such as the issue of the speed of the simulation which was found to be too slow for some and too fast for others. While these recommendations and evaluations relate to this specific system the issues and concerns are common to all learning environments based on qualitative or quantitative simulation.

Keywords: Learning Environment, Qualitative and quantitative models, evaluation, user feedback.

1. Introduction

This paper reports on a study of learners using CPRODS, a learning environment that encorporates 3 qualitative and 3 quantitative models of the same heat exchange system. The focus is on the learners' experiences of using the learning environment as gathered from analysis of questionnaire answers. The learning environment, CPRODS, uses the same underlying models as its predecessor MSPRODS. The qualitative models are an integral part of the teaching material within the domain, for further details of the instructional design see Sime (1998), and for result of studies of learning effectiveness see Sime (1995, 1996). The difference being that the models have been re-implemented in Visual Basic on a PC, the domain or instructional material remains the same but the user interface and the cognitive tools, to support the learners have been completely redesigned and re-implemented. The user interface has been greatly improved and crucially a number of learner support tools were introduced into the learning environment. These learner support tools, or cognitive tools, are intended to support the learning process and improve the learning experience (Lajoie 2000, Lajoie and Derry 1993). To assess the learning experience, all 36 participants in the study completed a questionnaire and expressed their views through multiple choice questions and open answer questions. Comments on any aspect of the system were encouraged. The intention was to determine the strengths and weaknesses of the cognitive support tools, and other aspects of the learning environment from the perspective of learners. Potentially, this information can be used to generate recommendations for modification to the design of the learning environment or for how to use of the learning environment to best effect.

This paper will focus on two issues:
1. What do learners think of qualitative models used in learning environments?
2. What do learners think about cognitive tools to support learning?

Subjective opinions gathered from evaluation questionnaires are reported.

The design of cognitive tools to support learning within a virtual laboratory or simulation based learning environment is a complex process that takes input from research in cognitive science on human learning and reasoning processes and combines that with theory and practice from instructional science. Like many design processes there is room for interpretation and artistic expression that can mean that the end result is not as useful as intended! While our understanding of human reasoning and learning processes has grown substantially in the last 20 years, there has been less progress in our understanding of teaching. Especially, on how teaching interacts with learning; we may be able to build a cognitive model of learning but do we really know how to interfere with that process (i.e. teach) to bring about improved learning. Yes, teachers know how to do this, but many years of attempting to build expert teaching systems (or intelligent tutoring systems) has produced unconvincing results. While we understand many of the differences between experts and novices (Chi et al 1988) we do not understand the difference between an excellent teacher and a poor teacher.

Perhaps to avoid this lack of information, research into authoring tools has found two avenues, the first takes a theoretical stance and attempts to work out some guidelines from first principles, i.e. from a theory of learning and instruction, e.g. Sime (1998). The second approach looks towards practice and gathers large libraries of instructional methods extracted from the practice of teachers, e.g. the Generic Tutoring Environment GTE (van Marcke 1998, Johnson and Sime 1998). There are problems with both of these approaches. Within educational research, a pragmatic approach based on learning effectiveness can be seen i.e. what works in practice? What leads to improvements in learning? What can we do to support learners? How can we improve the learning experience? What aspects of a learning environment assist learning and what hinders learning?

This paper will question whether qualitative models are useful in teaching, whether they are appreciated by learners, and whether cognitive support tools are useful and used in the manner expected by designers? One of the questions examined below is whether or not the cognitive support tools do actually support the learning of the learners in the ways expected by the designers? There is already evidence from van Joolingen (1993) to suggest that the hypothesis scratchpad is not as effective, in practice, as one might expect. Is this due to the implementation or due to the design

of the tool? Does the advice tool provide helpful information or not? Does the overview provide a sense of where you are in the learning material? It is intended to guide the learner and give a sense of progress through the learning environment. These are some of the questions this study attempts to answer by analysing data from evaluation questionnaires given to learners after using the system.

2. The CPRODS Learning Environment

2.1. The Learning Environment

The learning environment, CPRODS, is related to MSPRODS (Sime 1998) in that the underlying 6 qualitative and quantitative models are the same. There are 6, inter-related, models of the same physical system. The Bytronics Experimental Process Rig is a commercially produced heat exchange system that was designed for the laboratory teaching of theory and application of control theory. It is typical of a class of industrial heat exchangers and had been designed to be used in the teaching of undergraduate engineering students.

The 6 models represent different analyses of the Process Rig, each appropriate for solving a particular set of problems. There is a qualitative representation of the thermal processes in the system, a corresponding quantitative representation, qualitative and quantitative models of the flow around the system, and a qualitative and quantitative model of the complete system.

The models were chosen in consultation with an experienced teacher to illustrate key difficulties in understanding experienced by students during learning. The learning objective is to understand the relationship between heat and flow processes. In addition there are various concepts that are also learnt.

The learner is supported in the learning environment by several features, or learner support tools. These tools are designed to provide support to the cognitive processes of learning. The learner is given access to multiple simulations based on qualitative and quantitative models, a set of assignments, and a concept explanation facility. The assignment tool provides a set of assignments for the learner to do, there is a recommended order but the learners can select any assignment to do. The concept explanation facility provides information on basic concepts and terminology. In addition, CPRODS also contains 3 new tools: an advice feature, an overview, and an hypothesis scratchpad which are described in more detail below. The simulations have an improved interface over previous MSPRODS implementations and include representations of the trends. This is a feature that was in the first implementation and has been reinstated in this implementation. An outcome of the study will be to assess whether these representations of change in variables over time, are useful to learners.

The final assessment has been revealed to participant learners through the pre-test. The participants know the type of questions they will be asked and can therefore choose their investigations of the learning environment to meet this perceived need. Motivation is provided only by the learner's internal motivation to do well, the actual results will not influence their life in any way and so ultimately are unimportant. It would have been better if this had been integrated within an existing curriculum but this was not possible at the time.

2.2. The Learner Support Tools

There are five tools within the learning environment that provide support to the learners in their exploration of the simulated environments. For the purposes of the study two are considered part of the basic learning environment and the three others are referred to as cognitive support tools: the advice tool, that provided help on demand; the

overview tool that gives an overview of the learning material, cognitive tools and content; and the hypothesis scratchpad that assists in the forming of an hypothesis which can then be tested.

The Advice tool provided explanation of the concepts within the domain, e.g. the "dead band". Basic concepts within the learning material include: sensor, dead band, manipulated variable, controller, control system, set-point, settling time, error, feedback, closed-loop control system, open-loop control system.

The Overview tool provided a graphical representation of the knowledge in the domain with the assignments already attempted marked. This Overview provides a scaffold to the knowledge and also serves as a means of measuring progress though the learning material.

The Hypothesis scratchpad tool provided a set of pull down menus so that variables could be combined and the results of experimentation recorded. This enables the learner to record trials of different variables using a simulation and from the analysis of the data, extract the trend or relationship (Bierman et al 1990, de Jong et al 1994).

3. Method

3.1. Participants

36 undergraduate students were recruited (and paid a fee) to participate in the study of learning of learning with C-PRODS. The students were all aged in their early 20's, and were familiar with computers. Some had experience of computer based training. They were from a mixture of backgrounds some scientists, engineers and others were studying politics, education, psychology and other social sciences. 7 additional participants were used to pilot test the learning environment and the questionnaire.

3.2. Design

Two groups participated in the experiment, the first group used the basic learning environment and the second also had the benefit of the additional cognitive support tools. The additional cognitive tools enabled learners to request advice (Help), get an overview, or use a hypothesis scratchpad. Participants were given complete freedom in how they interacted with the environment. There were a number of assignments that could be undertaken but this was not compulsory. There was also access to explanations of concepts, and the qualitative and quantitative models that could be run so that they could observe and adjust the behaviour of the system. All participants, in both groups, were given the same time limit to interact with the learning environment. Each group were given an introduction to the use of the learning environment and the cognitive tools (for the second group only).

All participants were given a pre-test and a post-test, the pre-test established their existing level of knowledge in this field and the second test was used to determine their knowledge after use of the learning environment. The questions in the pre and post test were not identical but were matched so that they covered the same aspect of the domain and were of the same level of difficulty. Participants were also given an evaluation form to complete.

3.3. The Evaluation Form

The evaluation form asked the participants for their opinion of the learning environment, the learning materials, and the cognitive tools. The cognitive tools group answered 32 multiple choice questions and 16 open questions, a total

of 48 questions. The other group had the same evaluation form with questions about the cognitive tools removed resulting in an evaluation form with 33 questions including 23 multiple choice questions and 10 open questions. The multiple-choice questions asked participants to tick one of 5 boxes representing a five point scale, e.g.

1.b. Was the concept explanation window easy to use?				
Useless		Useful		Very Useful
			☑	

Open questions asked more general questions to determine whether participants were using the cognitive tools as expected, e.g.

3.d. What is the purpose of using the overview tool?……**I actually mostly used it for managing time: spend 1/3 in each field. I also told me that I had almost completed a field – so it was no use going on.** ……..

Evaluation questions were grouped around features of the learning environment:

1) The Concept Explanation window – 5 questions.
2) The Advice Window (Cognitive tools group only) – 5 questions.
3) The Overview Tool (Cognitive tools group only) – 5 questions.
4) The Hypothesis Scratchpad (Cognitive tools group only) – 5 questions.
5) The Assignment Window – 5 questions.
6) The Qualitative Models – 6 questions.
7) The Numerical Models – 6 questions.
8) General Questions
    9) freedom of exploration in the learning environment - 2 questions.
    10) Feedback, its clarity and frequency – 2 questions
    11) Duration and difficulty of the instructional materials – 2 questions
    12) difficulty and time to complete the pre-test – 2 questions
    13) difficulty and time to complete the post-test – 2 questions
14) Other Remarks – 1 question.

4. Results

4.1. How useful are the tools?

1.a. How *useful* was the concept explanation window?				
Useless		Useful		Very Useful
				☑

These grade were translated into numbers -2, -1, 0, 1, 2 and these are used below to indicate where on the scale was indicated by participants.

All the cognitive tools were rated at least “useful” (0) although some were considered more useful than others. The advice tool, overview tool, hypothesis scratchpad were considered useful while the concept tool, assignment tool, the qualitative model and the numerical model were considered “more useful” (1) but none rated “very useful” (2).

4.2. How easy to use are the tools?

1.b. Was the concept explanation window easy to use?				
Useless		Easy to use		Very Useful
			☑	

All the cognitive tools were rated at least moderately easy to use (0) with the exception of the hypothesis scratchpad which was rated (-1) a bit difficult. The easiest to use was the advice tool. Both the qualitative and the quantitative models were thought to be easy to use (1).

4.3. Which tools were worth using?

Participants were undecided about the value of the hypothesis scratchpad and the overview tool (0) but they thought the advice tool was worth using(1) and the concept, assignment and qualitative and quantitative models were definitely worth using(2).

4.4 Was it clear how to interpret the trends?

6.d. Was it clear how to interpret the trends (graphs of values over time)?				
Unclear		OK		Clear
			☑	

Questions 6d and 7d related to the clarity of the graphs for the qualitative and quantitative models. The results showed that on the whole participants thought that the clarity of the qualitative models was OK (0) and the numerical models were considered slightly better than OK (1). Most thought the graphs clear but 5 out of the 31 thought the numerical graphs were unclear and 7 thought the qualitative graphs were unclear. One reported that "I found the graphs confusing and didn't use them for the assignment." On the other hand some reported that the trend graphs were useful, e.g. "The graphs on the bottom were helpful because they gave a qualitative idea to go along with the quantitative (numerical)".

4.5. How much freedom in exploration was wanted?

When asked about freedom to explore the environment, participants suggested the current level was OK (0), however one reply was conditional "less given the time, with more time the current freedom is perfect" This indicates that perhaps the question was not specific enough about the learning objectives and time constraints.

4.6. Was enough feedback given?

When asked what they thought of the feedback and whether they wanted more or less feedback the participants were positive except for 2 persons who were negative. The result was an OK (0). One participant commented that the "the qualitative and numerical model graphics were a bit crowded and it was difficult (a little) to take in, at first."

4.7. Were the tools used as intended?

In general the tools were used as intended. Feedback from questions 1d, 2d, 3d, 4d, 5d, 6e, and 7e asked about the purpose of the tool and this highlighted when learners were not using the tool as it had been designed to be used. For

example, several participants only used the overview tool for time management, and as a means of measuring time on task. E.g. “I mostly use it (the overview tool) for managing time: spend 1/3 in each field. I also told me that I had almost completed a field - so it was no use going on. I totally forgot to use the relationship graph”. Another commented “I only used the % bar graphs at the bottom” of the overview screen “to monitor your own progress from time to time.”

4.8. What was seen as restrictive in the learning environment?

There were a variety of responses to this question. Common themes included the inability to abort from running a qualitative model, “Not being able to stop an assignment using a qualitative model until the correct settings had been achieved.” Also the difficulty of using the hypothesis scratchpad, “Too many options and complicated relationships forced my restricted use of this tool”. Thirdly, “I found the graphs confusing..”, “graphs difficult to interpret.” The graphs of trends of variables over time were found to be difficult to read.

5. Discussion

The discussion takes two parts, the first draws some recommendations out of the results of the evaluation of CPRODS, the second part examines Kolb’s theory of experiential learning and sees whether CPRODS provides all the elements necessary for learning by doing, according to Kolb (1984).

5.1. Recommendations

It is reassuring to see how easily the students accepted and dealt with the qualitative models. Some seemed to see the benefits of “rough calculations” while others saw the qualitative models as inferior, and less accurate, than the numerical models. This is perhaps due to prevalent perspectives on learning that places accuracy and detail high in importance.

This highlights the importance of presenting qualitative models to learners in a positive manner with plenty of explanation to the learners of the benefits of qualitative reasoning. This can either be presented as an intermediate step to quantitative understanding or as a useful tool for “rough calculations”. This is the first recommendation.

A common view is that learning is about searching for the finest grain of detail – this is a reductionist view of learning. One in which the basic aim of science is to reduce complex phenomena to separate simple parts and that this reduction provides explanation of the phenomena. While scientific understanding may benefit from this approach, it is not always an appropriate method of learning. It all depends on your learning objective. If the objective is to understand the components of the system then it may be appropriate, whereas if the objective is to understand the processes involved in the system, to control and manipulate its behaviour then a reductionist analysis is not the best approach.

The output from the simulations in the form of graphs showing change over time of variables was found to be clear to some and unclear to others. However, due to the varied group of students from a variety of disciplines, it may be that some are more familiar than others with interpretation of graphs. This generates a second **recommendation that the learners must be trained (either through existing skills, or additional training) in the interpretation of the simulation output.** This was backed up by one participants more general comment, “the qualitative and numerical model graphics were a bit crowded and it was difficult (a little) to take-in, at first”.

Many comments were made on the design of the learning environment that can be taken into account in further implementations. For example, when asked about the qualitative simulation…"some of the instructions were unclear. Weren't told what was happening while model ran before message came up to say assignment was achieved." The qualitative simulation which was found to be too slow by some participants and too difficult to control by others and they wanted a running commentary, rather than feedback just at the end of the assignment. Irritation was also expressed at "Not being able to stop an assignment using a qualitative model until the correct settings were achieved". The issue of speed is an interesting one common to many simulation based training systems. There are many situations in which learning to control a system in real time would take many hours whereas a training system can provide additional experience purely by speeding up time. In this system, the simulations were seen to be too slow, e.g. "Realistic but perhaps a little slow on reacting:" Other participants commented on how fast the qualitative models were. "it's difficult to watch everything at the same time, everything seemed to move along very quickly", "you had to run the (qualitative) models more than once in order to see everything that was going on". The models are actually quite realistic, perhaps even faster than the real system, but realism is not necessarily best for learning. This leads to a **recommendation that the speed of the simulation should be tested with learners during the design of a learning environment to take into account user tolerance as well as consideration for learning opportunities**. The simulation should not be so fast that it is difficult to see what is happening, nor so slow that it irritates.

5.2. Supporting experiential learning

In 1984 Kolb published a book describing his 4 stage model of learning by doing. In it he presented a cyclical model of experiential learning. The main points of his theory are:

1. that the learner is active in exploration,
2. the learner must reflect on the experience in a critical and selective way,
3. that the learners must be committed to the process of exploring and learning,
4. that there must be scope for the learner to achieve some independence from the teacher and
5. that the teacher imposes some structure on the learning process so that the learners are not left to discover by trial and error.
6. The learner must feel supported and encouraged
7. The trainer must provide appropriate learning activities and teaching methods to support each stage of Kolb's cycle (of experience, reflection, study the theory, plan the next experience).

Let's consider each of these points in turn and see how C-PRODS matches up to Kolb's model of experiential learning. (1) The qualitative and quantitative models within the learning environment give plenty scope for active exploration of the simulations. There was evidence of different individual learning styles. While some preferred to gain the basic concepts from the concept explanation window first, others preferred to explore. E.g. I felt that I did not use this option (the concept explanation window) as much as I should have tending to opt more for trial + error + seeking logical deductions." The second point will be addressed last.

(3) Commitment to the process of exploration and learning is a factor of motivation that I assume is dealt with outside the learning environment. However, feedback from the learners indicates a high degree of motivation and interest in learning in these students who were paid to take part in the study but for whom the learning means nothing. The learning itself is not part of their curriculum but many still managed to generate interest in learning and to enjoy the learning experience. For example, comments like "I enjoy the experience and learnt…", "Well worth undertaking again", "I found it an interesting task to do", "It was a pleasure to work on it", "I wanted to play more".

(4) It was possible for the learners to gain independence from the 'teacher' in that they did not have to follow the assignments they could do them in any order or ignore them and simply explore the simulations. This was a feature

that may not have been clearly conveyed to the learners as one reported that “I didn’t realise the assignments could be done in any order”. Others showed independence e.g. “I didn’t always follow the advice”.

(5) Structure was imposed through the provision of assignments that provided a structure to the learning this was backed up by: the advice tool that could suggest what to do next if the learner was stuck; the overview tool that gave an idea of the content and progression through the learning material; and the concept explanation tool that provided definitions and explained basic concepts and terminology. The concept tool and the assignment tool were available to all learners as part of the basic learning environment.

(6) Additional support was provided by the cognitive support tools, the hypothesis scratchpad, the advice tool and the overview tool. This was a very supportive environment with so many (five) tools provided. At the design stage there was concern that there were too many tools and therefore that the learning curve was too great for the learners who may be put off the learning environment. This may be true for some learners, as some reported that “I didn’t use this tool” even when it was available to them. This may be due to individual learning styles, differing levels of independence or concerns over time (which several participants expressed). Some did make comments like “It takes a long time to get into the environment, so there is little time to really enjoy the numerical models”.

(7) The learning environment provides a range of learning activities from free exploration, to hypothesis generation and testing, to learning about concepts from the concept explanations, to assignments that compare the behaviour of more than one model, to assignments that ask the learner to control the behaviour of the system. There are a variety of tasks that should suit a variety of learners’ individual learning styles. In hindsight it might have been better if learners were initially taught how to use the hypothesis scratchpad through some easy assignments. This may have increased the use of the hypothesis scratchpad that was considered the most difficult tools to use, but not the one least worth using – that was the overview tool. The learning environment enables the learner to study the theory (concept explanation tool), experience (free exploration of the simulations and control of their behaviour), reflect (generate and test hypotheses then reflect on the hypothesis tested, or use the overview tool to reflect on progress through the content) and plan (generate a hypothesis, or select the next assignment).

(2) The weakest link in this cycle is the reflective process, unlike many systems used in training in military and commercial areas, there is no after action review, or debrief of the training session. Reflection is supported only in that the learner reflects on the relationships within the field (supported by the hypothesis scratchpad) and reflects on progress through the content (using the overview tool), but the learner does not reflect on the learning process itself. Specific tools to reflect on the learning that has been achieved can be produced such as in the ETOILE project, (Dobson et al 2001, Pengelly et al 2000) where representations of the learner’s actions during training are recorded and presented back in a variety of graphical representations. These external representations are then used by trainers and trainees to discuss what happened during training and to reflect on the learning process.

6. Conclusions

This paper has reported some initial analyses of the questionnaire data from learners and has produced some recommendations for consideration in future simulation based learning environments. The learner support tools have also been assessed and all have been considered a positive force although some participants chose not to use them. This is their choice, of course. The hypothesis scratchpad was found to be the least useful and the least easy to use learner support tool but it was not the one considered least worth using, that was the overview tool. This may be due to the greater needs to learn to use the tool. The overview tool was not much used and many only used it to gauge progress over time rather than take advantage of all its features.

The qualitative models are readily accepted by learners and seen as worthwhile although more could be done to explain the reason for using 'rough calculations' in problem solving. Analysis of the data has revealed some interesting comments which can be fed back into the design process and generate modifications to the design and presentation of the learning environment, e.g. issue of the speed of the simulation which was found to be too slow for some and too fast for others.

The learning environment has been evaluated to see if it has all the features that Kolb (1984) says are necessary for experiential learning, or learning by doing. The five learner support tools do provide most of the learner support that Kolb prescribes, although reflection on the learning process is identified as a weakness of the learning environment. On the other hand the five support tools available to the learner are a major strength.

While these recommendations and evaluations relate to this specific system the issues and concerns are common to all learning environments based on qualitative or quantitative simulation.

7. Acknowledgements

This study would not exist without the hard work of Mr. Ken Korsmit who programmed CPRODS and who assisted in data gathering while he was a visiting student at CSALT. The study was financially supported by a grant from Lancaster University, Faculty of Social Sciences Sciences, Small Grants Scheme.

8. References

Bierman, D.J., Kamsteeg, P.A. & Sandberg, A.C. (1990). Student models, scratch-pads and simulation. In E. Costa (Ed.), New directions for intelligent tutoring systems (pp. 135-145). Berlin: Springer-Verlag.

Chi, M.T.H., Glaser, R. & Marr, M.J. (1988) The Nature of Expertise. Hillsdale, New Jersey: Lawrence Erlbaum Associates.

Dobson, M.W, Pengelly, M, Sime, J.A, Albaladejo, S, Garcia, E, Gonzalez, F. & Maseda, J. (2001) "Situated Learning with Co-operative Agent Simulations in Team Training" Computers in Human Behaviour. Volume 17, issue 5-6, on pages 547 - 573, cover date September-November 2001

Forbus, K.D. and Feltovich, P.J. (eds) (2001) Smart Machines in Education: The Coming Revolution in Educational Technology. MIT Press.

Johnson, R. & Sime, Julie-Ann (1998) *Authoring and GTE.* Instructional Science journal, special issue on GTE: A Generic Knowledge Based Tutoring Engine, guest edited by Kris van Marcke. Volume 26, Nos 3-4 July 1998, pp 227 - 242. Kluwer. ISSN: 0020 4277.

Jonassen, D.H. and Carr, C.S. (2000) Mindtools: Affording Multiple Knowledge Representations for Learning. Chapter 6 in S. Lajoie (ed) Computers as Cognitive Tools: No More Walls. Volume II. LEA: Mahwah, N.J.

de Jong, T, van Joolingen, W, Scott, D, de Hoog, R, Lapied, L & Valent, R. (1994). SMISLE: System for multimedia integrated learning environments. In T. de Jong & L. Sarti (Eds.), The design and production of multimedia and simulation-based learning material (133-165). The Netherlands: Kluwer Academic Publishers.

van Joolingen, W. (1993) Understanding and facilitating discovery learning in computer based simulation environments. PhD thesis, Eindhoven University, Netherlands. ISBN 90-386-0242-1

Kolb, D.A. (1984) Experiential Learning – Experience as the source of learning and development. N.J. Englewood Cliffs.

Lajoie, S. eds (2000) Computers as Cognitive Tools: No More Walls. Volume II. LEA: Mahwah, N.J.

Lajoie, S. and Derry, S. eds (1993) Computers as Cognitive Tools. Volume I. LEA: Hillsdale, N.J.

Van Marcke, K. (1998) *GTE: An epistemological approach to instructional modelling.* Instructional Science journal, special issue on GTE: A Generic Knowledge Based Tutoring Engine, guest edited by Kris van Marcke. Volume 26, Nos 3-4 July 1998. Pp 147-191. Kluwer. ISSN: 0020 4277.

Pengelly, M, Sime, Julie-Ann, & Dobson, M.W. (2000) Using Shared Mental Models as a basis for developing team competencies. In Proceedings of Workshop on advanced instructional design for complex safety critical & emergency training. Intelligent Tutoring Systems Conference (ITS2000), 19 – 23 June 2000, Montreal, Canada.

Sime, Julie-Ann (1995) *Model Progressions and Cognitive Flexibility Theory.* In J. Greer (ed) Artificial Intelligence in Education, 1995: Proceedings of AI-ED 95 - World Conference on Artificial Intelligence in Education. Washington, DC; August 16-19, 1995. Pp 493-500. AACE: Charlottesville, VA.

Sime, Julie-Ann (1996) *An Investigation into Teaching and Assessment of Qualitative Knowledge in Engineering*. In P. Brna, A. Paiva & J. Self (eds) European Conference on Artificial Intelligence in Education. 30 Sept. - 2 Oct. 1996, Lisbon, Portugal. Pp 240-246. ISBN: 972-8288-37-9

Sime, Julie-Ann (1998) *Model Switching in a Learning Environment based on Multiple Models.* Interactive Learning Environments journal, "Special issue on the Use of Qualitative Reasoning Techniques in Interactive Learning Environments", guest edited by Bert Bredeweg and Radboud Winkels. Volume 5, 1998, pp 109 - 124. Swets & Zeitlinger. ISSN: 1049 4820.

MM*for*TED: A COGNITIVE TOOL FOSTERING THE ACQUISITION OF CONCEPTUAL KNOWLEDGE ABOUT ARTEFACTS[1]

ELIO TOPPANO

Dipartimento di Matematica e Informatica
Università di Udine, Via delle Scienze 206, Loc. Rizzi, 33100 Udine, ITALY

Abstract. The work described in this paper deals with conceptual understanding of technical artefacts. A working hypothesis is that the comprehension of something by someone is strongly related to the capability of the subject to build and use multiple representations of the thing under consideration and to explicitly consider their characteristics and their reciprocal dependencies in explanation and problem solving. A cognitive tool called MMforTED has been developed to foster the acquisition of several conceptualisations (ontologies) that can be used to represent and inquiry about artefacts from different perspectives. The system exploits an instructional environment implemented by an electronic hypertext, that enables users to browse through a network of models of several devices. Browsing through the network of models is accompanied by a simultaneous change of perspective. Because of the particular organisation of domain knowledge we have adopted, this amounts to exercising a well defined knowledge transmutation (e.g. conceptual abstraction, generalisation, reduction, approximation and aggregation or their inverses). The ability to explore a situation from different conceptual perspectives is considered fundamental for problem setting and design development.

Key words: conceptual understanding, ontologies, multiple models, model transmutations.

1. Introduction

This paper is concerned with conceptual understanding of technical artefacts. Conceptual understanding means being able to reason about domain concepts and their relations. We assume that the core of undertanding is an hermeneutic activity of constructing interpretations [8]. The need for computational tools supporting understanding and interpretation has been stressed by various authors. Protzen et al. [17] write:

"Instead of computational problem solving attention should be given to the power of computational tools as communicative devices: devices that aid in the development of new and different understanding of problematic situations. It is this potential of computers to enhance our own understanding that needs to be explored"

In our opinion there are, at least, two main approaches for developing such kind of systems. The first approach is to build systems that substitute the human in making the interpretation. This approach has been followed, for example, by [10]. Another approach is to build cognitive tools that help the user acquiring the basic conceptualisations and relative knowledge transmutations that can be used by herself to make the interpretation. To this end, some authors, such as Laurillard [14] stresses the importance of systems that support tutorial dialogues; others, see for example, Ohlsson [16] suggests the analysis of *epistemic activities* (arguing, describing, explaining, predicting, etc.) which are considered more relevant for conceptual understanding than the study of goal oriented action or procedural skills.

The work described in this paper follows this second approach. The instructional system we are going to describe is aimed at fostering the acquisition of several *conceptualisations* that can be used to describe, explain and understand artefacts. The system, called MMforTED - an acronym for MultiModelling for Technical Education - has been conceived for introductory courses on technical education and design theory in the secondary school (age range 14-18). It exploits a learning environment, implemented as an hypermedia, constituted by a collection of *cases* of simple electrical and fluid mechanical devices. A case is represented by a

[1] This paper is a reduced and slightly modified version of a paper that will be presented at AID02.

graph of models [1] each one describing the artefact observed by a specific *conceptual perspective*. Models within a case are related together by codesignation links. Associative links are, instead, used to relate models of different artefacts featuring the same perspective. The collection of all available models and their interrelationships constitutes a kind of "conceptual landscape" that can be criss-crossed in many directions according to the type of problem or task to be executed (e.g. means-ends analysis, teleological explanation). Navigation through models includes:

- *browsing within a case*: the student can gain *cognitive flexibility* by being exposed to multiple interpretations (perspectives) of the same device [19]. Because of the particular organisation of domain knowledge we have adopted, changing perspective amounts to exercising a well defined knowledge transmutation;
- *browsing through cases*: the student can gain *knowledge transferability* by seeing multiple manifestations of the same interpretation.

Besides showing models of artefacts the system provides a scaffold of questions and ontologies upon which students can construct their interpretations of an artefact model together with exercises and meta knowledge about the instructional environment.

The paper is organised as follows. Sections 2 and 3 are devoted to describe the theoretical framework we have followed for representing artefacts from multiple perspectives. The framework leverages on the existence of a strictly layered organisation of conceptual knowledge. This organisation guides both the design of the educational material and the learning that subsequently takes place. In section 4 we illustrate the main characteristics of the instructional environment that has been developed to support conceptual understanding by showing how a typical page of the hyperspace is presented and which kinds of interactions are afforded to the learner. The section discusses also the experimental activity that is currently done with the system. Finally, section 5 draws conclusions.

2. The concept of model and model based communication

We assume here that a model is basically a device that is built to answer specific questions about some portion of reality. A symbolic model of an artefact is a description with the following properties:

- it is constituted by a set of *assertions* referring to the considered artefact;
- assertions describe the artefact in terms of: *entities*, *properties* of entities, and *relations* among entities;
- assertions are expressed in some *language* that has a well defined syntax and semantics and are externalised through a system of *signs*.

Notice, that for an object under investigation there is not "the model" to represent it but a set of models representing it from different points of view (i.e. using different types of entities, properties, and relations) or using different languages or systems of signs according to the observer's background knowledge and the kind of question or problem to be tackled.

Models can be used for problem solving or communication. This paper is mainly concerned with computer mediated communication of artefact models. We shall adopt a point of view that is strongly inspired to the FRISCO framework for information systems [5]. This point of view can be summarised as follows. In communication, a subject (the emitter) generates a message (e.g. a model) that represents some knowledge about an artefact expressed in a language. The message is transmitted via a channel (a medium such as a computer system) to another subject (the receiver) who interprets the message and constructs a personal conception of its content. Information is the personal knowledge increment of the receiver in interpreting the message. For FRISCO, information and communication are not absolute but relative concepts. They are seen as linking the individual person ("information" as increase of personal knowledge) and the larger community of which that person is a member (shared knowledge resulting from communication).

It is assumed that two main processes are involved in communication namely, interpretation and modelling. During *interpretation* the receiver perceives the message with her senses and forms a specific pattern of visual, auditory, or other sensations in her mind. These percepts are then elaborated by various cognitive processes such as categorisation, inference, imagination, etc. in order to form a mental conception. During *modelling* the emitter selects the content of a model that is, the aspects of a mental conception that are deemed relevant to answer the question or solve the problem of interest (articulation), and represents the content in a language and a system of signs (externalisation).

Interpretation and modelling are driven by the subject's conceptual system. The term *conceptual system* is intended here to refer to the collection of relatively stable conceptions (e.g. conceptual categories, cognitive models) formed in a person mind during her experience and interaction with the physical and socio-cultural environment in which she lives. In order for the process of communication to be effective it is necessary that the two partners in the communication process share a body of linguistic and conceptual knowledge about the domain of discourse that is they have to commit to an ontology.

3. Ontologies for reasoning about artefacts

In Artificial Intelligence, *ontology* is defined as "an explicit representation of a conceptualisation" [9]. Ontology is typically composed of two parts, that is, the conceptual-level ontology and the lexical level ontology. Lexical level ontology provides a human-friendly vocabulary of terms used by the authors of the ontology to describe the domain of interest. Conceptual level ontology specifies the detailed meaning of each concept, the relationships existing between concepts (e.g. taxonomic relations) and a set of semantic constraints (i.e. axioms). It is worth stressing the fact that an ontology in not only a specification of a conceptualisation but embodies an agreement about that conceptualisation. It helps people belonging to a community of practice or interest to identify what they agree on and what they don't about the domain of interest. Such agreement facilitates accurate and effective communication of meaning which, in turn, leads to other benefits such as inter-operativity, reuse and sharing [21]. In building the ontology for reasoning about artefacts we adopted an incremental design process keeping teachers of technical education in the loop. Teacher participation in ontology design has greatly enhanced practicality of the framework and its relevance to instruction.

3.1 The internal structure of the ontology: epistemological types and conceptual hierarchies

The proposed ontology is based on previous research on model based representation of physical systems [3]. According to this work, conceptual knowledge about artefacts have been partitioned into four categories called *epistemological types*. For each category a kernel collection of concepts and related terms together with a set of relevant *prototypical questions* and *problems* have been introduced. A brief description of epistemological types follows.

1. *Structural knowledge*. This type of knowledge describes which components constitute the artefact and how they are connected to each other (i.e. their adjacency). The basic ontology includes the concepts of "component", "terminal" and "connection". Structural knowledge can be used to represent the connectivity of a system;
2. *Behavioral knowledge*. This type of knowledge describes how components can work and interact in terms of the physical quantities that characterise their state and the physical laws that rule their operation. The basic ontology includes the concepts of "mode of operation", "physical quantity", "physical law", "behavioral state" and "trajectory" of states. Behavioral knowledge can be used for behavioral prediction, causal dependency analysis and sensitivity analysis.
3. *Teleological knowledge.* This type of knowledge describes the goals assigned to the artefact by its designer and the operational conditions which allow their achievement through correct operation. The basic ontology includes the concepts of "goal", "operational conditions", "use", "expected behavior". Teleological knowledge can be used for interpretation of actual use and definition of proper use.
4. *Functional knowledge* This type of knowledge describes the contribution of individual component behaviors to the realisation of the ultimate goals of the artefact. The concept of function is thus understood as a bridge between behavioral and teleological knowledge. For the class of artefacts whose behavior can be interpreted in terms of flow structures of generalised substances (e.g. material, energy, power, information) the bridge can be represented at two different levels of abstraction:
   Level 1: at this level the basic ontology includes the concepts of "generalised substance", "generalised current", "functional role" and "functional role network". Generalised substances represent the abstract entities that flow through a system while a current is the amount of generalised substance that flows through a unit surface in a time unit. The functional role of a component is an interpretation of its behavior - more precisely of the physical equations governing its behavior - aimed at characterising how the component contributes to the realisation of the flow structure in which it takes part. Examples of functional roles are: the conduit, the barrier, the reservoir, and the generator. A component may play different roles in different domains: in a flat-iron, for example, a resistor is a conduit of current in the electrical domain and a generator of heat in the thermal domain. It should be stressed that the association of a functional role to a component is done in a principled way by exploiting formal analogies between laws belonging to different physical domains [4]. Two types of relations between functional roles have been identified, namely, mutual dependency and influence. These are used to represent functional role networks.
   Level 2: at this level the basic ontology includes the concepts of "cofunction", "process", "phenomenon", and "functional organisation". Specific configurations of roles (called cofunctions) enable the occurrence of elementary processes such as transporting, reservoir charging, reservoir discharging and blocking. Elementary processes can be related together by specific relations such as direct causation,

regulation or support to generate phenomena. The network of processes (and phenomena) specifies the functional organisation of the artefact.

Functional knowledge can be used for functional prediction, functional dependency analysis and process detection.

Ontological proposals vary along the formality dimension from highly informal to rigorously formal ontologies [15]. The degree of formality required depends upon the intended purpose of the ontology. Since our purpose is communication an unambiguous but structured informal ontology (i.e. expressed in a restricted and structured form of natural language) has been considered sufficient. For the core terms of the ontology we use a definition template proposed by [21]. Figure 1 shows an example of concept definition. A critical issue in ontology design is the internal concept structure. Rather than having a single tree-like concept hierarchy the conceptual content of each epistemological type has been organised into a number of small local taxonomies including *type_of* hierarchies, *part_of* hierarchies and *precedence* hierarchies namely, *rank* hierarchies and *measure* hierarchies [11].

Term: TERMINAL
Short definition: a TERMINAL is a passive channel supporting possible interactions with the outside environment
Elaboration: a terminal supports just one kind of physical interaction which identifies its type
Example(s): thermal terminal, electrical terminal, mechanical terminal
Variations (synonyms): port, interface
Related terms/concepts: (type_of) structural concept, (part_of) component, (part_of) connection

Figure 1. Definition of the structural concept "TERMINAL"

Based on this organisation, five independent dimensions for conceptual variation has been identified. These are:

- *abstractness*: level in a rank hierarchy of epistemological types. The epistemological abstractness of a concept is a measure of the distance of this concept from the immediate experience within some theoretical framework. We consider the following ordering of abstractness: structural concepts are more concrete than behavioural concept which, in turn, are more concrete than functional and teleological ones.
- *generality*: i.e. level in a typological hierarchy of the conceptual entities and relations used to describe reality. For example, the type of entity "component" used within a structural model is more general than "electrical component" which, in turn, is more general than "resistor";
- *detail*: i.e. degree of granularity of the knowledge represented by a conceptualisation. For example, the structural model of an electronic device may include conceptual entities at the level of major subsystems (e.g. the concepts of "filter", "amplifier") or can be further refined at the level of elementary entities (e.g. the concepts of "resistor", "capacitor", "diode");
- *phenomenic coverage*: i.e. the range of phenomena taken into account by a conceptualisation and the kind of simplifying assumptions that it presupposes. For example, in order to represent the behaviour of an electrical circuit we can use the concepts of "voltage", "current", "resistance", the Ohm's law and the Kirkoff principles. Using these concepts it is possible to represent electrical conduction. However, the above conceptualisation prevents us to describe the effect of magnetic induction of the current flowing through a wire. Moreover, a physical phenomenon can be represented by means or more or less idealised laws. As an example, in representing electrical conduction in wires we can use the Ohm's law (i.e. V=RI) or we can abstract away the resistance R and use the idealized law (i.e. V=0);
- *resolution*: i.e. number of distinctions allowed by the domains of values associated to the attributes of entities and relations of a conceptualisation. For example, the resolution of a quantitative behavioural model can be lowered either by relaxing real valued variables and using qualitative domains of values such as the set D={negative, 0, positive} or by representing precise functional relationships by qualitative direct or indirect proportionalities.

Dimensions are used for multilevel representation of an artefact as it will be shown in the following sections.

3.2 The concept of perspective

The ontology provides the basic conceptualisations that can be used to describe a technical system. In our approach, each conceptualisation is represented by a *conceptual schema*. Formally, a conceptual schema CS is a tuple <*E, R, A/D*> where

- $E = \{E_i\}$ is a set of entity types;

- $R = \{R_j^k(E_{j1}, ..., E_{jk}\}$ is a set of relation types. Each relation type has a degree (k) that is the number of participating entity types;
- $A/D = \{A_j/D_j\}$ is a set of attributes representing general properties of entities or relations types. Each attribute (Ai) has an associated domain (Di) specifying the range of values the attribute may take.

Conceptual schemes are used, during articulation, to provide a semantic content to the specific entities and relations represented in a model. As a consequence the model can be seen as a set of entities that are instances of the concepts types specified by its associated schema .

In selecting the constituents of a conceptual schema we enforce the restriction that no concepts of the same schema can be taxonomically related. In other words we do not mix concepts having different abstractness, generality, detail, coverage and resolution in the same conceptualisation. This choice results in a rigid multilevel system of conceptual schemes each one representing a single perspective. By the term *perspective* we mean a specific choice of values (i.e. levels) along each of the above dimensions. Obviously, models inherit the perspective embodied in their schemes. It must be stressed that what can be seen from a given perspective is not "part" of the artefact but the whole artefact as can be perceived and conceptualised through its associated schema. Changing perspective means moving along one or more modelling dimensions thus changing the point of view (that is the conceptualisation) through which we interpret or represent reality. Because of the particular organisation of knowledge we have adopted this amounts to performing a specific type of *knowledge transmutation* as discussed in [20]. Table 1 summarises modelling dimensions, conceptual hierarchies and associated knowledge transmutations.

3.3 Representing artefacts using multiple perspectives.

Figure 2 shows an example of representation using multiple perspectives. The considered artefact is a simple lighting system composed by a battery that supplies power to a light bulb when a switch is closed. The representation includes seven models of the device: a structural model (M7); two behavioural models (a quantitative model, M5, and a qualitative - causal - one, M6); a functional role model (M4), a functional process model (M3) and two teleological models (M1 and M2) at different levels of detail. We use a diagrammatic language to represent the models. Links between pairs of models specify the *codesignation relations* existing among elements belonging to two or more descriptions of the artefact. For example, the link between the structural model M7 and the behavioural model M5 specifies which physical quantities and laws in the behavioural model correspond to which terminals and components in the structural one; the link between the behavioural model M5 and the functional model M4 describes which physical laws in M5 are associated to which functional roles in M4, while the link between the functional role model (M4) and the process model (M3) specifies the correspondences existing between cofunctions (i.e. specific configurations of roles) and processes.

TABLE 1. Basic knowledge transmutations and related modelling dimensions

Modelling dimension	**Conceptual hierarchy**	**Knowledge transmutation (and inverse)**
Conceptual abstractness	Rank hierarchy of epistemological types	Conceptual abstraction (Concretion)
Resolution	Measure hierarchy	Relation or value abstraction (Concretion)
Generality	Type hierarchy	Generalisation (Specialisation)
Phenomenic coverage (range of phenomena that are explicitly represented)	Rank hierarchy of idealised laws	Reduction (Expansion)
Phenomenic coverage (accuracy of relations used to represent relevant phenomena)	Rank hierarchy of idealised laws	Approximation (Elaboration)
Detail	Part hierarchy	Aggregation (Refinement)

Finally, the link between function and teleology is realised by associating goals in the teleological description with the phenomena (or processes) represented in the functional representation which are used to achieve them. The relation between processes and goals is, in general, many-to-many since a process may participate to the realisation of several goals and, conversely, a goal can be fulfilled by utilising several processes. Codesignation relations can be used to switch from a description to another in order to focalise reasoning or disseminate information.

4. The MMforTED Prototype System

MMforTED is a cognitive tool. By this term we intend an instrument that can support, represent or perform an identifiable cognitive process that is part of the complete learning experience by a learner [22]. The target cognitive processes are multiperspective analysis and multilevel reasoning. *Multiperspective analysis* means to be able to analyse and conceptualise an artefact from different perspectives and to be able to map corresponding elements belonging to different points of view. *Multilevel reasoning* means to be able to integrate several points of view and representations to solve problems and to be able to change representation by selecting the aspects of the object under consideration that are more relevant and the level of accuracy, resolution, and detail of the descriptions that are deemed more appropriate for the problem to be solved. Figure 3 shows how the learner may access the information space. A typical page is divided into areas that are distinguished by different colour backgrounds.

• *Reasoning with a model*: this is the main presentation area (A). Each device model is displayed in this area both diagrammatically using a plex structure and by a short textual description to facilitate interpretation and remembering. This area includes major buttons for analysing the model:

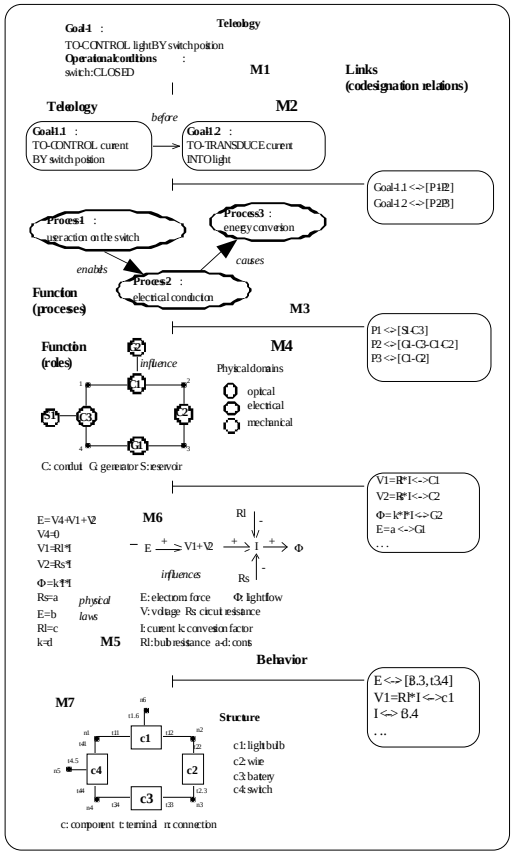


Figure 2. Representing a lighting system using multiple models

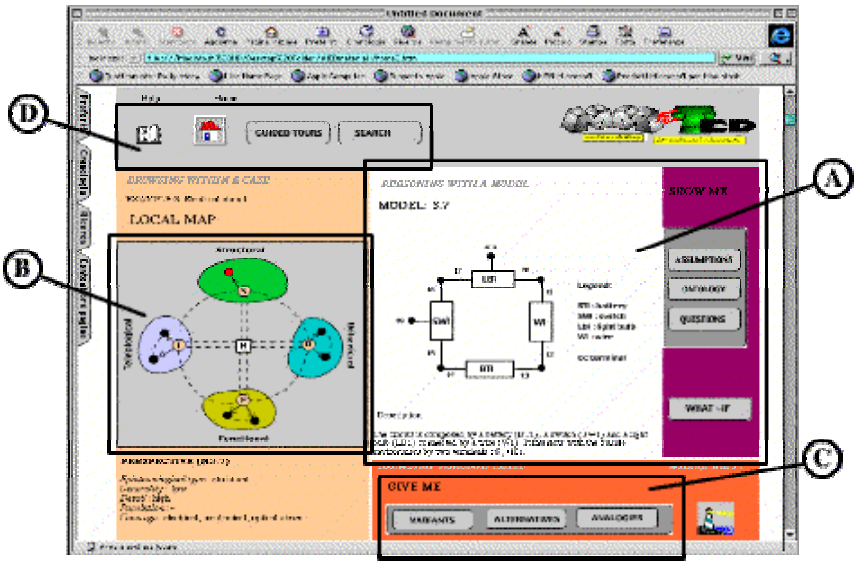


Figure 3. The page representing the structural model of a lighting system

- Assumptions: visualises the main modelling and operational assumptions taken by the modeller to build the description.
- Ontology: describes the ontology (e.g. the conceptual schema) used to build the description.
- Questions: provides examples of prototypical questions for reasoning about artefacts using different perspectives. Questions are linked to ontologies and to a set of exercises that are simple quizzes aimed at testing the ability of the learner to discriminate among epistemological types.
- What-if: proposes possible changes of the assumptions lying behind a model in order to analyse their effects both within a perspective and across multiple perspectives. For instance, by clicking on the "What if" button the student may select a different operating mode for the artefact or hypothesise an abnormal state for a component and observe the effects of the modification. Actually, the selection is made within a set of predefined modifications.

• *Browsing within a case*: this is the primary navigation area (B). It contains a clickable map to browse within a case. The map shows the available models of the artefact under consideration which are clustered according to their epistemological type. By clicking on the arc connecting a pair of models it is possible to visualise relational knowledge i.e. codesignation links. This is valuable to connect model fragments (snippets) through perspectives in order to understand the behaviour and functioning of specific subsystems or component assemblies.

• *Browsing through cases*: this is the secondary navigation area (C). It contains several buttons for browsing through cases:

- Alternatives: proposes a list of models, across cases, which share the same ontology (hence the same perspective) of the currently displayed one.
- Variants: proposes a list of cases which have the same purpose (goal) of the currently displayed one but different functional organisations.
- Analogies: proposes a list of examples which share the same *type* of functional organisation but the organisation has been realised by exploiting different physical phenomena.
- Where am I: shows actual position (case) in the global environment of available cases.

Finally, the area at the top of the page (D) includes general services, namely: help, home, guided tours and search.

MMforTED has been designed to be usable for diverse learning goals and thematic threads so we do not impose a single overarching perspecive or global narrative struture to the material. The idea is to let the students develop their own comprehension by exploring the information space of cases and observing the many ways artefacts may be described and the many ways the general principles behind their functioning become manifest. This is in accordance with modern learning theories such as constructivistic theory that points out the positive effects of letting the learner create her understanding and knowledge structures. Moreover, the student is requested to

reflect on the structuring of knowledge presented through the environment and on the ontologies and questions that each model calls forth in order to make hypotheses about the behaviour and functioning of the artefacts.
An experimental activity with the system is currently done within the ICARO project a national initiative whose principal aim is to provide the discipline of technical education with an epistemological foundation and to produce a set of guidelines to help teachers develop educational curricula and web-based educational material. The project involves a selected set of schools and teachers, several regional educational research institutions and experts from different fields such as pedagogy and engineering sciences. Teachers have been fully involved in the design and evaluation of the system as well as in planning the activity to be done with the students. This activity has three distinct phases:

- *Pre-assessment*: students are asked to write an essay where they describe a given artefact and explain how the artefact achieves its purpose. The goal of this activity is to pre-assess the student's capability to understand the artefact functioning and to use appropriate concepts and terminology in describing her understanding.
- *Reflective learning*: The teacher introduces the multiperspective approach and models how to deal with the MMforTED environment. Then she fades her involvement while coaching and supporting the students in their own navigation.
- *Post-assessment*: finally, the student is invited to write an essay where she describes the behaviour and functioning of a given artefact (which is different from that used in pre-assessment). The result is then compared with the essay generated before the experimentation.

In the post assessment phase, students are forced to abandon their status of “hypertext audience” and are engaged in being teachers of the meanings they have created by exploring the learning environment and reflecting on the extent and quality of their knowledge. Evaluating their learning experience involves evaluating them as designers of models. These models are then compared in terms of consistency, terminological appropriateness and completeness with respect to epistemological types and perspectives. At the current stage of the experimental activity, we can only provide a qualitative assessment. While many of the pre-assessment essays are characterised by limited generation of concepts, poor terminology and a general inability to describe the object under consideration from several conceptual perspectives, we find that all students involved in the experimentation, each one at his or her own pace, made improvements in their skill to interpret and argue about the given artefacts. We are actually exploring the role of collaboration in improving such results. The idea is that if the students have to collaborate to write the final essay then they are forced to explicitly externalise their conceptions and points of view, understand each other and negotiate meanings. The knowledge negotiation approach in Education holds that the goal of education is not knowledge acquisition per se, but to acquire the flexibility to participate in the discourses of several communities of practice, that is specific groups of professionals acting and communicating in specific ways. Participating in professional groups implies the ability to understand the important debates and problems and use the right language and conceptualisations to examine and influence ongoing debate.

5. Conclusions

In the last decade several intelligent systems for instruction or training in technical domains have been proposed. Most of them are designed to support the acquisition of procedural knowledge or skills. For instance, XAIDA [12] is a system for the development of computer based maintenance training, Cycle Pad [7] supports modelling and simulation of thermodynamic systems; the Science Learning Space [13] is an inquiry learning environment designed to support learners in performing discovery skills. MMforTED does not directly support any specific activity, but is focused on the acquisition of the conceptual knowledge that is propedeutic to perform complex activities such as, for example, discovery, diagnosis, design. In discovery learning , learners construct their own knowledge by experimenting with a domain and inferring rules from the results of these experiments. Because of this constructive activity, it is assumed that students will understand the domain at a higher level than when the necessary information is just presented by a teacher or an expository environment. In practive, as discussed in [22], it has been very hard to find solid evidence for this hypothesis. Our position is that students need more than just the domain to learn about it. Apart from access to domain information (e.g. cases) they need assistance in selecting and interpreting this information in order to build their knowledge bases. In fact, making hypotheses, during an inquiry process, involves the ability to formulate questions about the object under consideration. In particular if we consider an hypothesis as a guessed relationship among a set of entities (i.e. Rk(ei, ...ek)) than the search space of hypotheses that can be formulated by the student critically depends on the student's knowledge about the possible relationships that can be assessed and the types of entities that can be involved in a relationship. Hence, our claim is that, if the student has been exposed to a space of possible interpretation then she can enrich her vocabulary of entities and relationships and, thus, enlarge her search space for hypothesis

construction. MMforTED seems to meet general acceptance. One reason is that it is perceived as an elaboration of an actual practice: the RARECO method [6]. This is an heuristic method widely used in the first two years of the secondary school to support the construction of knowledge about artefacts and the production of technical texts. The method is constituted by four phases, which are sequentially performed by the students under the supervision of the instructor:

1. **R**epresentation. In this phase, the artefact under consideration is represented by a picture or by a realistic drawing. The main components of the artefact are then highlighted and labelled.
2. **A**nalysis. For each component a definition is provided together with a specification of its purpose and physical properties;
3. **RE**lation. The students are encouraged to identify the main processes occurring in the artefact and to relate them together. For each process, the components that take part in the process are specified.
4. **CO**munication. The above analysis is translated into a text describing the structural decomposition of the artefact and its functioning.

The main contributions of our system with respect to RARECO are: i) the ability to support a richer variety of perspectives including structural, behavioural, functional and teleological models as well as all major dimensions for model variation proposed in AI literature (i.e., detail, coverage, value and relation abstraction, resolution, epistemological abstractness); ii) a particular attention to conceptual knowledge (i.e. ontologies) and model based reasoning; iii) the use of the web as the content provider as well as the delivery medium of instruction.

Actually, models have to be built by hand. This is a difficult and error prone process. We are exploring the possibility of using plex replacement production rules with applicability conditions to specify transformations between models [20]. As a final remark, we think that the system could be useful also to the teacher who can browse the information space in order to select materials (e.g. models, exercises) to be presented off line i.e. in a traditional class lesson. The modularity resulting from the organisation of domain knowledge allows the teacher to identify different types of problems; to build exercises in a more focalised way in order to exercise specific capabilities (e.g. abstraction, generalisation, aggregation, etc.) or satisfy particular educational objectives. Moreover, the proposed knowledge organisation and the use of perspectives allow one to derive an order of presentation of domain knowledge: from "simple" to "complex" models. To this end, the teacher has several degrees of freedom: she may select, for example, on the base of epistemological type or according to the level of generality, detail, resolution, coverage, etc. of modelling concepts. Therefore, it is possible to build a progression of learning experiences [23] that can be used to support a competence based approach to instruction.

References

[1] Addanki, S., Cremonini, R., and Penberthy, J.S., 1991, Graphs of Models, *Artificial Intelligence* **51**, 145-177.

[2] Andriessen, J. and Sandberg, J.: 1999, Where is Education Heading and How about AI?, *IJAIED*, **10**, 130-150.

[3] Chittaro, L., Guida, G., Tasso, C. and Toppano, E.: 1993, Functional and teleological knowledge in the Multimodelling approach for reasoning about physical systems: a case study in diagnosis, *IEEE Trans. on Systems, Man, and Cybernetics*, **23**(6), 1718-1751.

[4] Chittaro,L., Tasso, C. and Toppano, E.: 1994, Putting functional knowledge on firmer ground, *Applied Artificial Intelligence*, **8**(2), 239-258.

[5] Falkenberg, E.D., et al.: 1998, FRISCO - A Framework of Information System Concepts - *The FRISCO Report IFIP WG 8.1 Task Group FRISCO*, Web version: ftp://ftp.leidenuniv.nl/pub/rul/fri-full.zip.

[6] Famiglietti Secchi, M.: 1984, Laboratorio Tecnico, IGDA, Novara.

[7] Forbus K.D., Whalley P.B., Everett, J., Ureel, L. Brokowski, M., Baher, J., and Kuehne, S: 1999, CyclePad: An articulate virtual laboratory for engineering thermodynamics. *Artificial Intelligence* **114**, 297-347.

[8] Gadamer, H.: 1976, *Philosophical hermeneutics*, Berkeley, CA, University of California Press.

[9] Gruber, T.R.: 1993, A translation approach to portable ontology specification, *Knowledge Acquisition*, **5**(2), pp.199-220.

[10] Haymaker, J., Ackermann, E. and Fischer, M.: 2000, Meaning mediating mechanism, *in* J.S.Gero (ed.), *Artificial Intelligence in Design '00*, Kluwer Academic Publishers, pp. 691-715.

[11] Hieb, M.R. and Michalski, R.S.: 1993, Multitype inference in multistrategy task-adaptive learning: dynamic Interlaced Hierarchies, *in* R.S. Michalski and G. Tecuci (Eds.), *Proc. 2nd Int. Workshop on Multistrategy Learning*, Harpers Ferry, West Virginia, pp.3-17.

[12] Hsieh, P., Halff, H.M., Redfield, C.L.: 1999. Four Easy Pieces: Development Systems for Knowledge-Based Generative Instruction, IJAIED, **10**, 1-45.

[13] Koedinger, K.R., Suthers, D., Forbus K.D.: 1999. Component-Based Construction of a Science Learning Space, IJAIED, **10**, pp.292-313.
[14] Laurillard, D.: 1993, *Rethinking university teaching*, London, Routledge.
[15] Noy, N.F. and Hafner, C.D.: 1997, The State of the Art in Ontology Design. A survey and Comparative Review, *AI Magazine*, **18**(3), 53-74.
[16] Ohlsson, S.: 1993, Learning to do and learning to understand: a lesson and a challenge for cognitive modelling, *in* P. Reimann and H. Spada (eds.), *Learning in humans and machines*, Oxford, Pergamon Press, pp. 37-62.
[17] Protzen, J., Harris, D. and Cavallin, H.: 2000, Limited computation, unlimited design, *in* J.S.Gero (ed.), *Artificial Intelligence in Design '00*, Kluwer Academic Publishers, pp. 43-52.
[18] Rosenman, M.A. and Gero, J. S.: 1996, Modelling multiple views of design objects in a collaborative CAD environment, *INCIT'96 Proceedings*, pp. 49-61.
[19] Spiro, R.: 1997, Knowledge acquisition for application-cognitive flexibility and transfer in complex content domains, *in* B.K. Britton and S.M. Glynn (Eds*.), Executive Control Processes in Reading*, Erlbaum, Hillsdale, pp. 177-199.
[20] Toppano, E.: 1999, Using graph transformations to support multilevel reasoning in engineering design, *Machine Graphics & Vision*, **8**(3), pp. 395-425.
[21] Uschold, M.:1998, Knowledge level modelling: concepts and terminology, *The Knowledge Engineering Review*, **13**, 5-29.
[22] van Joolingen,W.: 1999, Cognitive tools for discovery learning, *IJAIED*, **10**, 385-397.
[23] White B., Frederiksen, J.: 1990, Causal model progression as a foundation for intelligent learning, *Artificial Intelligence* **42**, 99-155.

AN APPROACH TO TEACNING ENGIEERING DESIGN USING MULTIPLE PERSPECTIVE AND INTEGRATED PRODUCT MODELS AND SIMULATION

Xiu-Tian Yan

Department of Design, Manufacture and Engineering Management
Strathclyde University
James Weir Building
75 Montrose Street, Glasgow G1 1XJ
e-mail address: x.yan@cad.strath.ac.uk

Engineering design education is traditionally structured such that each individual discipline is taught by a specialist in that discipline. This has artificially created an impression to students that a product should be looked at from different perspective in a segmented manner. This approach hence results in a great problem when computer model based teaching methods are introduced to tackle multidisciplinary product design classes. There are clear gaps in student's knowledge as well as excuses that they have forgot what they have learnt. On the other hand, the competitive market requires better products designed and made in a shorter time with a higher quality and at a lower cost. The speed of launching products onto the market becomes an increasingly important factor for a new product to become successful. With rapid advancement in computer technology development both in terms of hardware capability and software functionalities, engineering designers as well as product designers are better equipped to create and make new and novel products than ever before. However many engineering graduates are not taught effectively to use these technologies to benefit their future design activities. In addition, these computer tools are not fully used to explore their potentials in teaching and learning.

This paper presents the practical experience of using model based learning approach in an education subject area - product development, which is an area of great complexity and diversification. It describes a broadre view of using model based learning approach in the education of new generation of engineering designers by introducing an innovative design model for computer based engineering design. The approach is based on an integrated and coherent use of several existing computer aided design (CAD) tools. Through the use of these tools as learning support systems, students are trained to be aware of multi-perspective models and their integration during a product development. Students learn in-depth knowledge of each subject area by using each aspect models. At the same time, by integrating these models, they can appreciate integrated design approach. Knowledge representation of product engineering design is also discussed in the paper. The paper highlights the current practice in product design engineering education and the problem areas for model based system and qualitative reasoning for intelligent tutoring system. An example of how these systems have been used is described in the paper to illustrate the benefits of the approach. As part of continuing research effort, the paper also describes a proposed outline framework to develop a suitable intelligent learning/tutoring system to improve the effective learning of such a complex teaching topic.

KEYWORDS

Model based teaching, multi-perspective modeling, knowledge intensive tutoring, and design education.

1. INTRODUCTION

The increasing pressure from customers in today's competitive market requires many manufacturing companies to produce a variety of products to satisfy rapidly changing market conditions, which include both functional requirements as well as aesthetic requirements fro different culture and time. In addition, rapid advancement in technology development means that there are now far more options available to a designer to solve a given design problem than any other time in design history. Companies as well as education organizations hence actively seek for enabling technologies to facilitate their product development and improve the productivity and effectiveness in generating new design solutions and products. In order to achieve the above, designers want to make more informative design decisions to cut down rework and to be more confidence about their design solutions and the intended product performance at the end of design process.

Due to human being's limitation in remembering relatively small amount of information, it is infeasible to expect a designer to remember all abstract description and details of a product being designed – a product model. Instead, a designer should utilize his/her strength in judging a solution based on multiple criteria reasoning and previous experience, making decisions and knowing how to perform a number of design studies. By taking the above division of tasks, storing information and retrieving details of partial design solutions can be separated from these overall high-level tasks. This naturally leads to the identification of computer systems to support design, as computers are good at remembering and processing a vast amount of information associated with a product being designed. Computer systems can also handle very well both static and dynamic information being generated during design processes.

Over last decade, the computational power of computers has been improved enormously. At the same time computers are available with rapidly reduced cost relative to performance. It is now possible to rethink the traditional design methods and practice taking consideration of these rapid design environment changes. At the same, due the complex nature of product design process and modelling, no such a computer has been developed, which is really powerful enough to handle all aspects of complex product modelling and design. This is therefore an interesting time for engineering designers to use these tools as they face the dilemma of having some support, but not all support they hope for. Therefore the methods of making student as well as practical design engineers aware of how to use these tools to their best advantage become very important in a successful product development.

More importantly for product design engineers is the fact that a number of advanced Computer Aided Design (CAD) systems are now available at affordable prices and can be run PC-based platforms. The use of advanced CAD system in the product development has added a new dimension in maximizing the use of these systems. This has created great opportunities for engineering designers especially working in small to medium sized companies, to think how to make best use of these available technologies.

Computer based tutoring system to address this need at both the detailed level of using one particular CAD system and at the high level of identifying suitable CAD system functions is of great important in educating next generation product design engineers.

This paper describes a proposed approach to cover this very wide yet important area of education based on the idea of using computer multi-perspective product models to teach how to support product modelling and simulation in the product development. It focuses on the real needs of developing a practical approach for product design engineers to use advanced CAD tools. This approach is based on the research results of using integrated computer based design approach in product design and experience that the author has had in practicing this approach in teaching various classes for both BEng/MEng Product Design Engineering and MSc in Computer Aided Engineering Design. The design environment and facility available to students involved in practicing this approach are very similar to many small to medium sized companies. Using these environments to model a product from a multi-perspective point of view and represented in multi-complexity level is the proposed pragmatic approach to improving design solution generation and evaluation process. This approach allows one to investigate more design alternative solutions (improve solution quantity at conceptual design stage) and produce better-considered solutions (improve final solution quality). Potential difficulties that one might face using this approach are also discussed in the paper. Examples of student projects will be given to discuss what can be achieved by applying the techniques introduced in the class in their design projects. Finally an outline of a proposed computer aided learning/tutoring system will be described to show the direction of future research work and the system development.

2. PRODUCT DESIGN AND MODELLING

Product design engineering is a subject of true multi-discipline nature, and the focus of the design activities is on product development rather than on a specific traditional academic discipline. It is a well-known fact that many products under consideration by designers are often the artefacts engineered using knowledge and technology from a number of disciplines. The design research community has been studying and tried to understand the engineering design and its process at least or the last half century. The common understanding of engineering design so far is that design is a process of generating solutions, which satisfy customer's requirements (French, 1985). A number of design process models have been since created, represented by French's model, Pahl &Beitz' model (Pahl &Beitz, 1996)and Pugh's model(Pugh 1991). These models are intended to be general and aim to guide designers to go through a series of stages and carry out a number of design activities in order to understand and solve design problems. However these design process models don't provide sufficient and specific guidelines for product development using computer based tools. It is therefore difficult to use them in a computer support design environment. In an effort of clarifying a design process model for product modelling and simulation using computer support tools, the author has derived a computer support design process model shown in Figure 1. This model is based on the research work carried out on the deployment of several computer support systems. These systems include FORESEE (Borg et al 2000) and FORESEE 2 (Yan at el 2001), DeCoSolver (Sawada and Yan 2001), Schembuilder (Bracewell 1995).

The design process can be broadly divided into three stages in a computer supported design environment, namely the design problem understanding through an analysis of need, the initial solution generation through the conceptual design, and the solution refinement and finalization through the embodiment and detail design. So far in the development of computer support for engineering design, there is little well developed support for the first two stages of the design process, mainly due to the complexity and diversification of these design activities during these stages. The final stage of the design is currently the main area of reasonable computer support, which can be used to aid engineering designers to improve their design. This stage of computer support can be further decomposed broadly into component modelling, component matching and sizing, and behaviour simulation and comparison for informative decision making. This decomposition enables one to investigate even further the constituents of each of these design support activities. It is argued in this paper that the conventional component geometry-based modelling should be enriched and broadened to be multi-perspective modelling for a product. Based on this enrichment of modelling, t is therefore possible for the subsequent behaviour evaluation and comparison to be also multi-perspective. Only with such a design support environment, can a designer have a full support for a thorough evaluation of any solutions. During any design process, designers also need to use reference information e.g. working principles, component database etc., as shown in Figure 1 to support a designer to be more productive, systematic and effective. In addition, design information in Figure 1 has the following feature. Design information tends to be qualitative and abstract at the early design stage and this information become more quantitative at the later design stage as more and more design decisions are committed to concretise a design solution. This design information feature is ideally suited for computer-based support as computer systems support well incremental expansion of deign information. Since the current computer support systems have not been fully developed to support multi-aspect modelling and simulation, a pragmatic framework and concept in thinking of using currently available systems are important in this approach. In addition, skills and techniques are required to fully support multi-perspective modelling and simulation. The following sections of this paper describe an advanced way of using modern computer systems in supporting product design and development. The roles of computer based simulation tools, the benefits and precautions of using these systems are also discussed in the paper.

3. MULTIPLE PERSPECTIVE MODELLING

A product is an integral artifact consisting of many facets or aspects, which require consideration and engineering before it becomes a marketable commodity. Most of current education system educates an

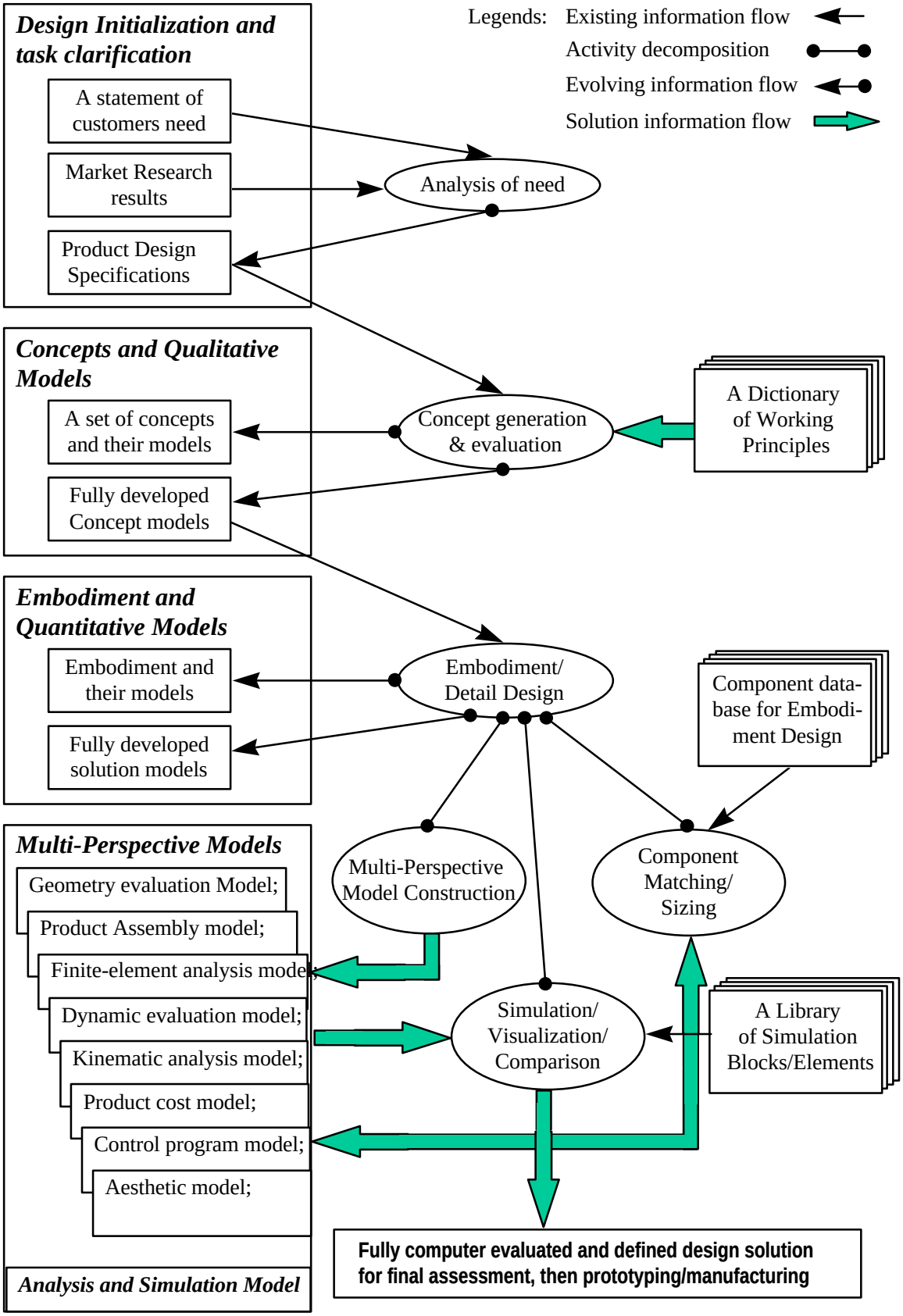


Figure 1. A proposed framework to support learning of design process and its modelling and simulation for product design

engineering designers in such a way that a student design engineer studies each aspect of important perspectives of product development in a segmented way. Little emphasis has been put on the integral aspect of all these important perspective of product development. The results of the education system are that students can work out a given problem in a reasonable depth. But they lack training in integrating various aspects of product development in an effective manner. They tend to think the other aspect of the product development is not their responsibility. There is a big gap between these type of graduate design engineers and what is expected, especially from small to medium sized companies, as they can't afford to a so many specialists. Even for the large organizations, specialists don't always communicate with each other effectively as they don't normally share a common language.

To be successful in today's competitive market, manufacturing companies need to produce more innovative products as well as a variety of products to satisfy the consumers' changing needs. This requires engineering designers know how to use more efficient and effective methods of conceiving more design alternative solutions, selecting suitable concepts from this long list of different concepts, refining chosen design solutions, and finally converting them into manufacturable products to satisfy ever-increasing and rapidly changing market needs. One important approach to address the above need is to make engineering designers to be aware of possible consequences associated with chosen design solutions. By making best use of the computer-based design tools, the product development lead-time can be reduced and quality of a design solution can be improved through more coherent and integrated use of these tools. From previous research work (Yan and Sharp, 1994, Yan 1992)), it is demonstrated that multi-perspective product modelling and simulation can help product design engineering practitioners to advance the design practice by producing better considered product design solutions with little increase of the product development cost. Figure 1 shows the important modelling aspects that one should consider from a number of perspectives during the produce development. These include: the geometrical modelling, the kinematic modelling, the assembly modelling, various analysis modelling including Finite-Element Analysis modelling, product dynamic behavior modelling, product cost modelling, control/control programming modelling, and ergonomic/aesthetic modelling. It is obviously difficult to find an all-round perfect system which is able to handle all the above. The key to the multi-perspective modelling proposed in this research is through the product model partition and integration. An appropriate model partition using product perspective views as guideline will allow one to concentrate on the local modelling of an aspect of a product that a designer is interested in. The identification of these important aspects for a particular design problem concerned helps a designer to concentrate on the key aspects of the design problem. More effort and time spent on these aspects can ensure that better quality design solutions be produced with consideration of these aspects.

Within each of these important aspects, a concept of modelling a product using multi-model complexity level can be introduced to accommodate the requirements of modelling different details with different modelling resolution. More complex models can be created by including more details of an aspect of the product. These complex models enable a designer to conduct more in-depth investigation of a chosen design aspect. This can lead to a better understanding of the design problem, hence better solutions generated.

A complete product can be decomposed into sub-systems, and the modelling of the product can be tackled by creating sub-models of these sub-systems. The integration of these sub-system models at the abstract level allows one to have an overview of the product. This high-level overall model allows a designer to see the overall behaviour of the intended product to be derived from this solution model. During a product modelling, the process of generating, analyzing and evaluating a design solution using computer modelling should be fully supported. It is essential that a computer design support tool will be able to assist designers to focus on one aspect of a product in a great depth without loosing the overall sight of the product. Equally important, a designer should also be supported to have a good overall understanding of product models, without loosing the grasp to details of aspect models. It is therefore argued in this paper that the multi-complexity level and multi-aspect model based system to the product development is essential to facilitate designers to investigate a product's behaviour from a different perspective at a different level of the modelling resolution. This approach can also provide a compromised solution to the conflicting behavioural requirements of computational speed and level of accuracy of the model. By combining different sub-system models with different model complexity levels, a product model can represent a product with detailed representation of aspect/subsystem of interest to a designer. At the same this model will not require undue computational power to solve/display the model.

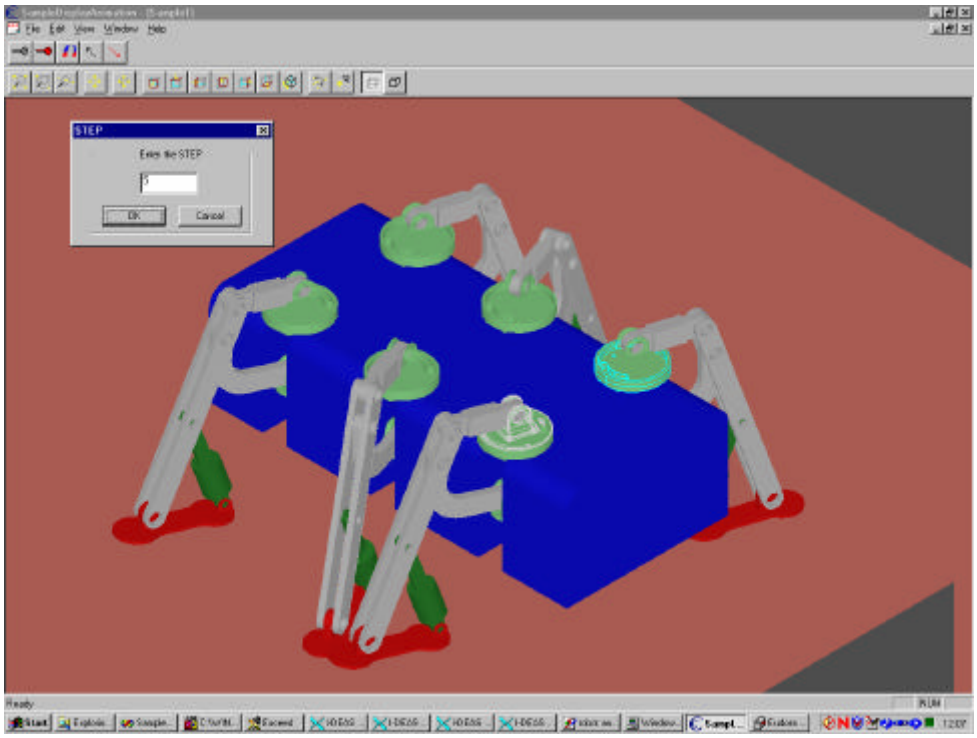


Figure 2 A modular wall climbing robot built from library modules and developed by a project student;

A tutoring system, which address the above model based learning system requirements, would help a learner to master these complex yet important concepts and methodologies. It is clear from the above discussion that the effective support for engineering design involves multi-perspective and complexity-level modelling of a product. This requires integration of product partial models from different perspective at different level of resolution. The data communication among these partial solution models, possibly through communication links using standard data formats/protocols, is the key to product model integration. To achieve an effective model composition, configuration and integration, a model library of the commonly used product modelling components should be and has been created. A collection of these models in each aspect forms a comprehensive reusable product-modelling library. Such a library provides a designer with a rich source of models to support multi-complexity level modelling and multi-perspective modelling. The interchangeability of the models at a different complexity level can be supported in this approach. Figure 2 shows an example product model – a modular wall climbing robot modeled and designed by two project students. The model used a number of stardard modules generalized in a robot-modelling library. This particular model was created to facilitate learning and understanding of such a robot's behaviour during its normal operation. It acts as a good learning tool to study the robot.

Models in Figure 3 were created using IDEAS and AutoLisp – a programming method provided by AutoCAD system to develop customized model and functionality. These models derived from the AutoLisp programming method give a designer/student much more to learn in terms of flexibility in manipulating these models, checking clearance etc. than a static model. Figure 3.c shows another example - a gear mesh created using AutoLisp program. Students learn much more effectively through this model-based approach. They can used fully developed geometry models to explore and understand what a mechanism does and how it behaves at the initial study stage in their early year study. They have to understand the basic principles of mechanisms before they can implement these mechanisms in a computer model in their later year study. Next section will discuss how simulation facility can be used to help students to learn design and modelling of products.

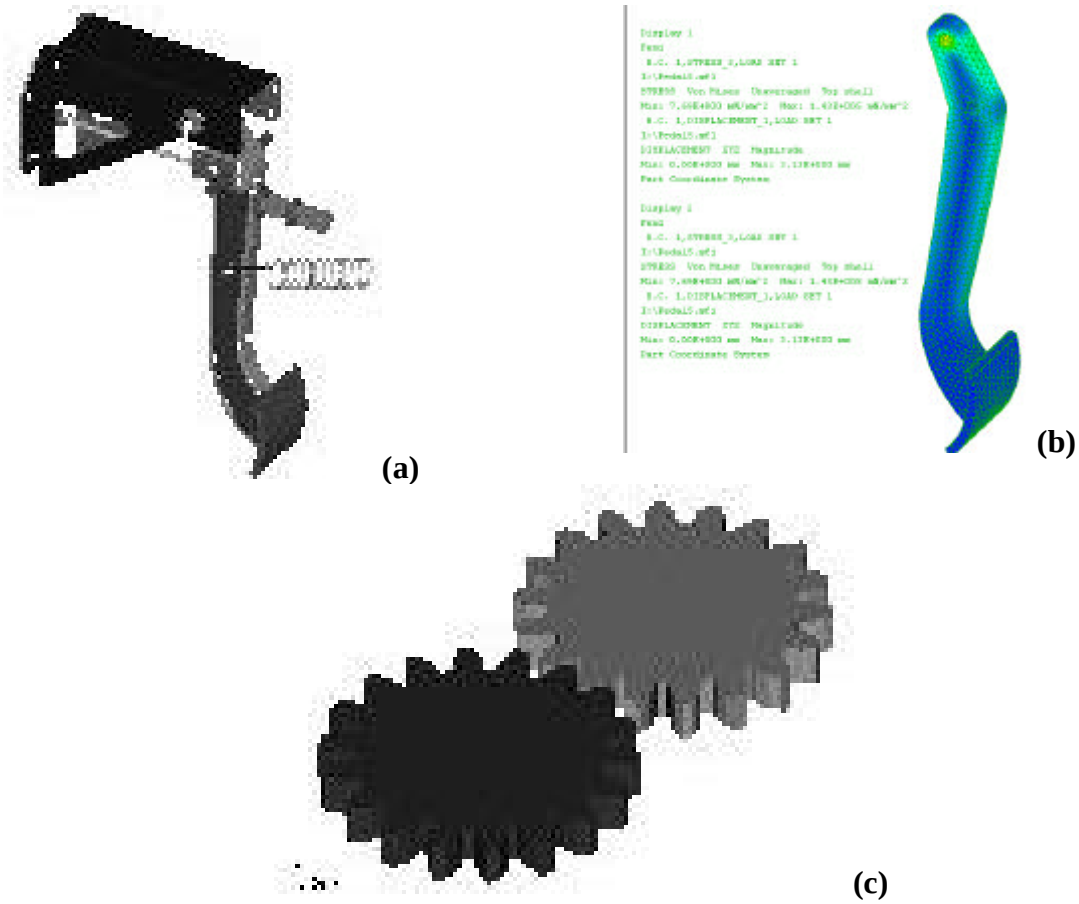


Figure 3 Some example model based learning models: (a) A geometry and its assembly model of a car brake system; (b) The FEA model of its key component, showing the stress; (c) A gear mesh geometry model and mechanism model for simulation;

4. MULTIPLE PERSPECTIVE SIMULATION

Based on the models described in last section, it is useful if students can see the behaviour of a physical system in a computer so that one can learn these physical systems without having them built. This can be done through the use of simulation techniques developed over last decade or so. Computer model based simulation uses embedded relationships between various parts of the system and can exhibit a correct motion, kinematic, dynamic and structural and thermal deformation behaviour of a physical system. Compared with a real physical system, each of the above aspect can be studied in a computer model separately so that a multi-perspective model based learning approach and its associated framework can be implemented to give an integral study of a product. Simulation technology has been traditionally used for system analysis and control system design. It has also been used for mechanism analysis and validation. Various simulation techniques have been studied and successfully used in different applications to predict the behaviour of a physical system. The notation and representation of a physical system can be different depending on the modelling methodology employed for a particular modelling approach. Typical modelling methods for simulation include *clock-based mechanism geometric solid modelling* (Yan 1992), *block diagram* (Gayakwad and Sokoloff 1988), *signal flow diagram*, *bond graph* approach (Karnopp et al. 1990), *Yourdon diagram* (Cooling 1990) and *schematic diagram* and so forth.

These simulation methods were generally developed for a specific discipline/aspect of a product and can only cope with the aspect of a product/system within the discipline/aspect. Whilst the simulation technology advanced in last two decades rapidly due to the significant computer technology advancements, it has only been used in the product design in a very limited way. Emerging interdisciplinary subjects, such as mechatronics, on one hand remove many design constraints and a designer can work in a much wider design space, on the other hand design process become more complicated due to much more possible combinational

solutions for a given problem. The verification of a product design scheme from multi-perspective point of view can become a quite difficult task without an appropriate computer support.

To overcome these difficulties, the author has developed a pragmatic approach to modeling these interdisciplinary systems. This approach establishes design project requiring multi-perspective modelling and simulation to enable product design engineers to understand fully design multidiscipline problems, and evaluate their solutions models.

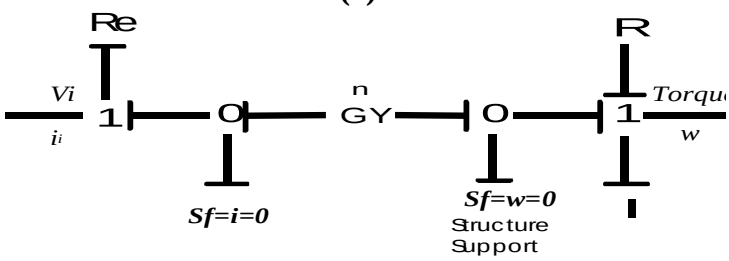
One of the obvious problems with many current computer modelling methods is that they are very much domain dependent. Product design engineers have to be trained to be able to use them in different application domains in order to be competent in using all these domain dependent technologies. This task itself is a very challenging task for product design engineers. In addition, when it comes to interfacing different application domains using corresponding modelling methods, modelling using a computer is becoming a daunting task even for many researchers, let along the new product design engineers. However, recently the modelling separation of data and control and function decomposition employed in Yourdon method suggests that a high level function block oriented method used to model high energy systems can also be used to model information related systems. This method suggested an integrated modelling approach adopted in this research and teaching could lead to advancement in modelling and simulation of interdisciplinary subjects such as mechatronic products.

5. A MULTI-PESPECTIVE LEARNING EXAMPLE

The role of modelling and simulation in product embodiment design is clearly shown in figure 1 where these methods can assist a designer to evaluate a scheme generated at conceptual design stage. Dynamic performance of a mechatronic product can be evaluated by using a unified simulation approach (Yan 1994),

(a)

(b)



(c)

(d)

Figure 4 An examples of multi-perspective product modelling. (a) A wheelchair geometry model, (b) A wheelchair assembly model for simulation; (c) A Bond Graph model for a D.C. motor; (d) Physical model of an intelligent wheelchair built by students

in which energy transformation from an energy form to another will be clearly illustrated and the efficiency of each component/sub-assembly can be readily available to a designer. The same computer model can be used to match the components, which interface with each other. This allows a designer to rapidly determine the correct parameter values for a given design scheme and carry out evaluation. The simulation system also provides a high interactive interface, allowing a designer to instantly change the values of any design parameter and get immediate feedback of the effect due to the changes made. This has proven to be an extremely useful facility, especially for new designers who need to repeat many time to learn and evaluate the behaviour of a design solution.

From the design aspect point of view, it can be and has been used to evaluate mechanism kinematics, which include the velocity, path and geometries of many mechanisms. In assembly modelling, computer can generate many frames of graphic representations of a product at different positions during its assembly process and these frames of images can then be animated. Similar process can be used to simulate the mechanism to visualize the tack/path of a particular point of a mechanism. An example used here is the design of an intelligent wheelchair. Both the assembly of the wheelchair and its mechanism has been animated as shown by captured screen dumps in Figure 4. This enables a designer to clearly identify the problem of a design by graphically simulating the kinematic and assembly behaviour of the product. A design leaner can visually learn and study these behaviors of the wheelchair. They also allow a learner to study the effect of modifying certain design parameters and be optimized rapidly by re-simulating the behaviour with a modified design. These models can also be used to study the optimization of design. This model based teaching and learning approach has been quite successful in teaching product design at advanced study level.

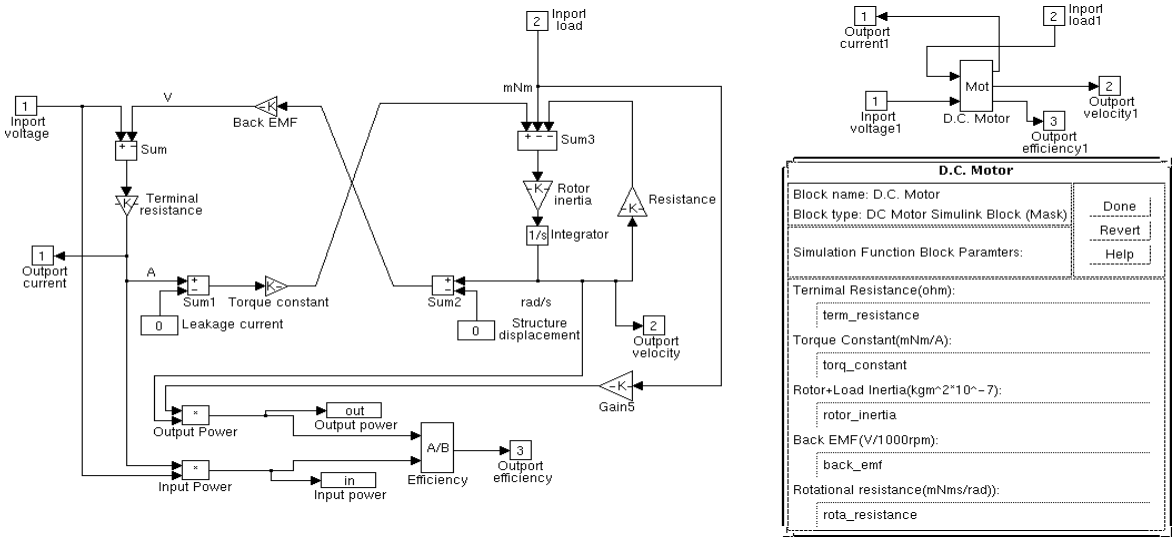


Figure 5 A simulation model of a D.C Motor used in the wheelchair project – left: the detailed model of the motor; right: a high level model masked into a block.

The other important perspective of product investigation is the dynamic simulation of an energetic system in which energy flow is of significant importance. For example, in the same wheelchair design project, d.c. motors and gear boxes are used to convert electrical energy to mechanical rotation energy to drive the wheelchair. A good understanding and appropriate selection of these energy components are vital to a successful wheelchair development. A model-based system developed by the author has been quite useful for students to learn these energetic aspects of the system. It allows students to study the energy flow and conversion from one domain to next. It also helps designers to have more insights regarding the underlying relationships between important design parameters. Students learn effectively how to design and select the correct power components based on the simulation models. Simulink has been used to enable students to simulate energy flow and control requirements. Figure 3.c shows a basic Bond graph model of a d.c. motor and its use in the form of converted model object is shown as in Figure 4.a. using Bond Graph theory based rules, a bond graph representation as shown in Figure 4(c) can be converted into a block diagram based representation, shown in Figure 5. This block diagram can be implemented in a simulation system such as Simulink, which allows one to model a dynamic system using control blocks. Using object-oriented software design methodology, a low-level complex block diagram can be further simplified into a high level block and

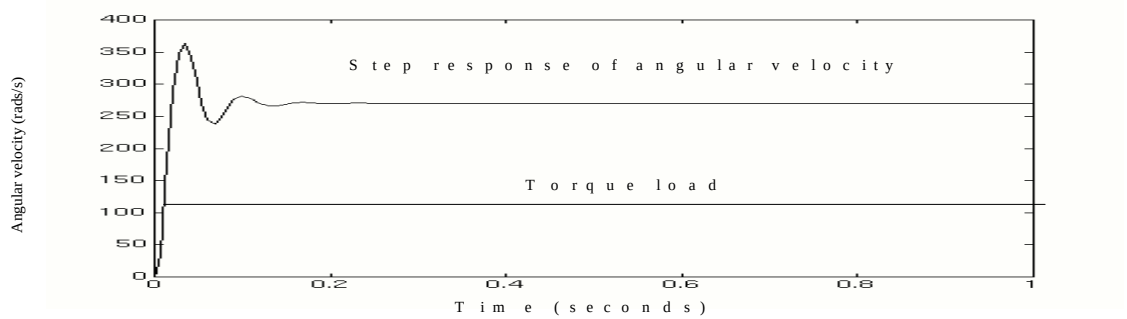


Figure 6 Simulation results of dynamic behaviour of one drive wheel of the wheelchair

Figure 5 shows a parameterized high level block representing a D.C. motor. With a library of such parameterized commonly used components, the dynamic performance of a mechatronic product can be evaluated by using a unified simulation approach developed by the author (Yan and Sharp 1994), in which energy transformation from one energy form to another will be clearly illustrated. A library of commonly used engineering components has been developed and is composed of a number of expandable sub-libraries. A user can easily select a component and configure a product model to simulate the behaviour of the product. The same computer model can be used to match the components interfacing with each other. This allows a designer to determine rapidly the correct parameter values for a given design scheme and carry out evaluation. The simulation system also provides a highly interactive interface, allowing a designer to instantly change the values of any design parameter and learn quickly the effect due to the changes made. This has proven an extremely useful learning tool offered to students, who need to repeat the evaluation many times to learn the behaviour of a design object. Figure 6 shows an example plot of simulation results of the dynamic behaviour of one drive system for the wheelchair. A designer can determine from these simulation results the maximum power requirements and maximum speed the wheelchair can travel and so forth. This information can help students to learn and gain great insights to the systems they are dealing with.

6. FRAMEWORK OF A MULTI-PESPECTIVE MODEL BASED LEARNING SYSTEM

Having described a broad product multi-perspective model based learning approach, this section gives details of a proposal to a more integrated and intelligent computer aided learning system, by incorporating some reasoning mechanism to facilitate the learning of product engineering design. It aims to develop a single learning system by bring all aspects discussed in this paper as well as other important learning and tutoring issues learned from other researchers. Model based diagnosis techniques discussed in other work [Koning et al 1999] will be used to provide more proactive support to learners. Figure 7 shows the architecture of the system. The architecture consists of four distinctive yet closely related modules. The learner user interface acts as an important medium for a learner to interact with the system and learn many techniques how to perform a task and concepts/principles of product modelling. All learner interaction with the system will be recorded into a user learning log file. In the opposite direction of information flow, identified suitable knowledge, explanations to a query and some possible suggestions to the learner about suitable actions the learner should take before moving on.

Based on the analysis of these logged learner actions, a reasoning mechanism can then determine the profile of the learner, e.g. level of knowledge of each of the subject area, competence of using the system, interest of modules of the subject area. etc. Also within this reasoning module, it can retrive relevant information to a learner's query, making suggestions or asking questions based on the learning behaviour up to this moment of time. Assessment questions can also be generated using a database within which there exist a number of predefined questions and learning stimulus at a suitable level of learner's ability. The reasoning modules can also derive learning advices or actions the learner should take once the learner the system detects the difficulties the learner is experiencing. The third module basically contains important subject knowledge models for product design engineering education. And the final module contains a number of database to provide information on generating learning stimulus proactively, learning assessment methods and associated questions, and learning advice and actions for learners having difficulty in mastering the subjects as well as using the system.

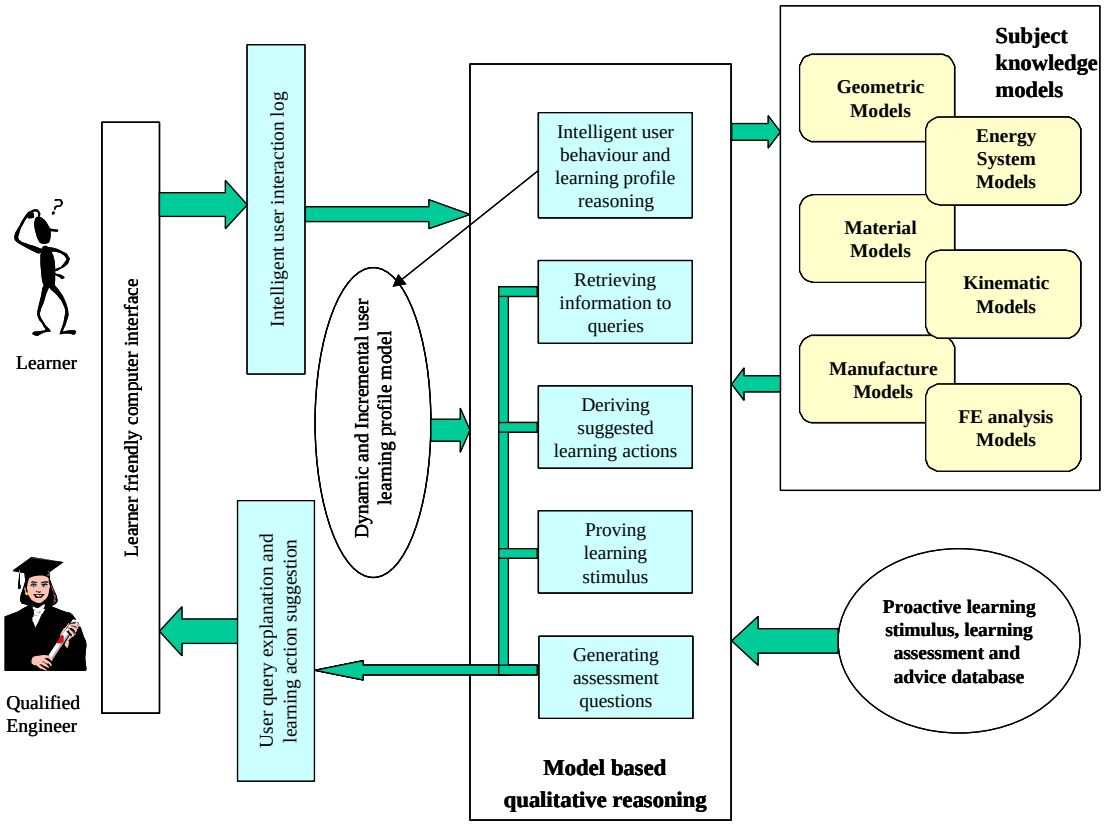


Figure 7 A proposed architecture of an multi-perspective product design learning system

7. CONLUSION AND FURTHER WORK

This paper has described a multi-perspective model based learning approach used by the author to teach multi-disciplinary product engineering design classes. A unified modelling approach, based on Bond graph and block diagram methods as underlying knowledge representation, has been developed and used in this approach. A novel function block oriented modelling method has been derived to support effective learning and proved effective in handling energetic system design. In combining the commercially available kinematic and assembly modelling and simulation systems through integration techniques, a multi-perspective model based learning approach has been developed. Through the introduction to and use of this learning approach, new designers such as senior student design engineers appreciated the importance of and gained the benefits of multi-perspective product design by using multiple models. This approach has broadened design engineers understanding significantly from single geometric modelling to much more comprehensive modelling and simulation in an integrated manner. However, from evaluation of the approach, design engineers found it is difficult to understand Bond graph theory, as student designers have not gained sufficient experience and knowledge about specific components they need to use. Industrial designers have not been introduced to this method either. This raised the need of developing higher-level encapsulated modelling method to help these designer engineers to adopt this approach.

In addition, the paper also described a framework for developing a single tutoring system based on the current teaching practice. Successful multi-perspective model based learning approach will be used in the system. In addition, qualitative reasoning and other means of reasoning and methods will be deployed to enhance the intelligent behaviour of the system.

The approach described in the is paper represents the practical part of the teaching approach adopted by the author. There are a number of areas for further development and research. The work has largely been tried on the education of mechatronic product design. The learning models generated in the component library are hence very much for mechatronic products. This library requires further expansion to cope with other domain specific product design. Currently, this approach is being used to design a mechanical product and it is hoped

through further investigation that the approach will be further evaluated. More importantly a single stand-alone system will be developed by incorporating all best practice in model based qualitative reasoning and other reasoning techniques, so that a true intelligent learning system can be fully developed for product design engineering education.

REFERENCES

Bracewell, R.H., Chaplin, R.V., Langdon, P., Li, M. Oh, V, , Sharpe, J.E.E. and Yan, X.T. 1995 (in alphabetic order), “Integrated Platform for AI Support of Complex Design(Part 1) and (Part 2)”, in AI System Support for Conceptual Design ed. by J. Sharpe, International Workshop on Engineering Design, Ambleside, England, March 27-29,.

Borg, J, Yan, X. T. and Juster, N. P. "Exploring decision’s influence on life-cycle performance to aid “design for Multi-X”", Artificial Intelligence for Engineering Design, Analysis and Manufacturing Cambridge University Press, Vol 14, pp 91-113, 2000.

Cooling, J.E., "Software Design for Real-time Systems", Chapman and Hall, 1990.

Cunningham, J. J., Dixon, J. R., (1988), "Designing with Features: The Origin of Features"; *Proceedings of the ASME International Computers in Engineering Conference*, Vol. 1, San Diego, pp. 237-243.

French, M.J. 1985, "Conceptual design for engineers", Springer-Verlag, second edition.

Horváth, I., Kulcsár, P., and Thernesz, V., 1994, "A Uniform Approach to Handling of Feature-Objects in an Advanced CAD System", in *Advances in Design Automation*, DE-Vol. 69-1, ed. by Gilmore, B. J., Hoeltzel, D. A., Dutta, D., Eschenaurer, H. A., ASME, New York, pp. 547-562.

Gayakwad, R. and Sokoloff, L., "Analogue and Digital Control Systems", Published by Prentice-Hall Inc. International Editions, 1988.

Karnopp, D.C, Margolis, D.L. and Rosenberg, R.C., 1990, "System Dynamics, A Unified Approach", Second Edition, John Wiley & Sons, Inc..

Kees de Koning, Bert Bredeweg _, Joost Breuker, Bob Wielinga 2000, “Model-based reasoning about learner behaviour”, Artificial Intelligence 117 (2000) 173–229

Pahl, G. and Beitz, W., 1996 “Engineering Design – A Systematic Approach”, Springer.

Pugh, Stuart, 1991 “Total Design – Integrated Methods for Successful Product Engineering”, Addison-Wesley Pub.

Sawada H. and Yan, X. T., “Preliminary Design Support System Based on a Generic Under-Constraint Solving Technique”, ASME 2000 International Design Engineering Technical Conferences and the Computers and Information in Engineering Conference - ASME Design Automation, Baltimore, USA, September 2000.

Yan, X. T. Borg, J. and Juster, N P 2001 “Concurrent modelling of components and realization systems to support proactive design for manufacture/assembly”, Journal of Engineering Manufacture, Proceedings of the Institution of Mechanical Engineers Part B, Vol215 pp1135-1141, September 2001.

Yan, X. T. 1997, “The Role of Simulation Tools in the Teaching of Product Design”, Proceedings of 11th International Conference on Engineering Design ICED97 Tampere, Vol. 3 pp. 469-472, ISBN 951-722-788-4, August 19-21,

Yan, X.T. and Sharpe, J. 1994, “Unified Dynamic Mixed Mode Simulation of Mechatronic Product Design Schemes”, Int. Workshop on Computer Aided Engineering Design, England, pp. 259-280.

Yan, X. T. 1992, “Graphic Modelling of Modular Machines”, Ph.D thesis, Department of Manufacturing Engineering, Loughborough University of Technology, UK,.