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

Abstract

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.

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.

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

	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.

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).

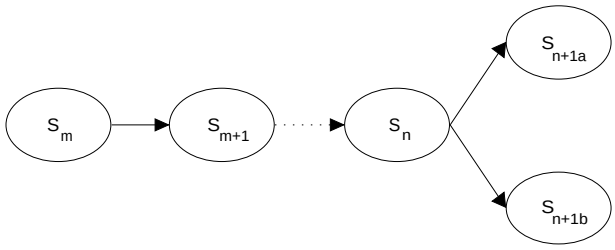


Figure 2. Path segments vs. paths. The path from S_m to S_n is also a path segment, but the path from S_m to S_{n+1a} (or S_{n+1b}) is not, because it includes a branching point.

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

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

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]

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 $\rightarrow$ 4 $\rightarrow$ 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 $\rightarrow$ 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 $\rightarrow$ 16 $\rightarrow$ 18 and 15 $\rightarrow$ 19 $\rightarrow$ 18. If these different paths contain the same events

[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.

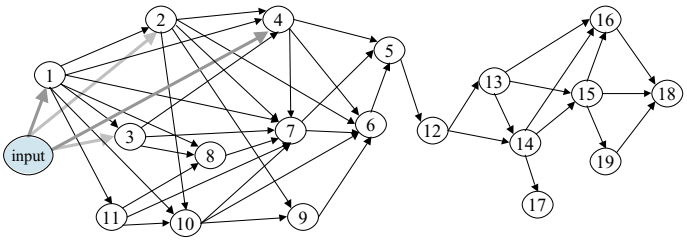
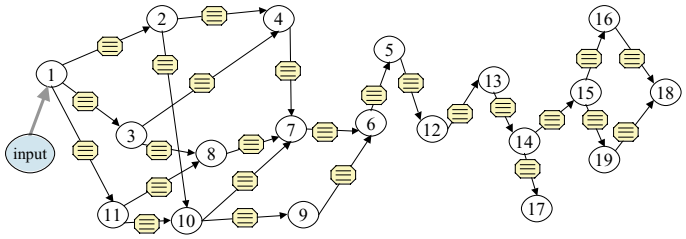
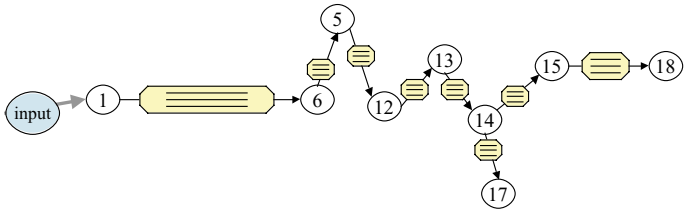
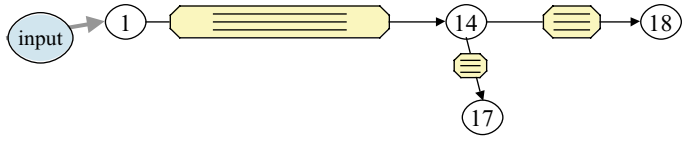
	The original state-transition graph as output from the simulator, consisting of 19 states, 43 transitions, and 896 behaviour paths. The first step in the aggregation process actually adds more information, namely the differences between states. The second step is performing *transitive reduction* (see 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

(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.