Learning by building:
A visual modelling language for psychology students

Paul Mulholland and Stuart Watt

Knowledge Media Institute

The Open University

Milton Keynes, MK7 6AA, UK

P.Mulholland@open.ac.uk and S.N.K.Watt@open.ac.uk

http://kmi.open.ac.uk/projects/hank

ABSTRACT

Cognitive modelling involves building computational models of psychological theories in order to learn more about them, and is a major research area allied to psychology and artificial intelligence. The main problem is that few psychology students have previous programming experience. The course lecturer can avoid the problem by presenting the area only in general terms. This leaves the process of building and testing models, which is central to the methodology, an unknown. Alternatively, students can be introduced to one of the existing cognitive modelling languages, though this can easily be overwhelming, hindering rather than helping their understanding. Our solution was to design and build a programming language for the intended population. The result is Hank, a visual cognitive modelling language for the psychologist. Our informal analyses have investigated the effectiveness of Hank in its intended context of use, both as a paper and pencil exercise for individuals, and as a computer based project to be carried out in groups. The findings largely support the Hank design decisions, and illuminate many of the challenges inherent in designing a programming language for an educational purpose.

1. Introduction

Most psychology undergraduates will at some stage study cognitive modelling (alternatively referred to as computational modelling or artificial intelligence). Eysenck and Keane [1] claim that a valuable development in cognitive psychology is the growing tendency to use both cognitive modelling and conventional empirical techniques to support theoretical developments in cognitive psychology. It is, therefore, important for psychologists to have some understanding of the research methods of cognitive modelling. Psychology students, though, are not expected to be programmers. Some cognitive modelling courses may consider just the underlying theory and major historical developments of the area. Others, have attempted to give psychology students first hand experience of cognitive modelling (e.g. [2, 3]).

Within this tradition, the cognitive psychology course at the Open University has included a cognitive modelling component since the late seventies [4]. For the first decade, the course used the SOLO language [5]. SOLO had limited functionality but was within the grasp of the non-programming psychology student, though it could only be used to develop a limited number of models. For the past decade the course has used the Prolog programming language, which is far more powerful than SOLO, but also far more difficult to use and understand (e.g. [6]).

The languages Prolog and SOLO represent two diverse design decisions within the usability-expressiveness trade-off [7]. Prolog is highly expressive (i.e. a powerful language) though it has low usability, particularly for the novice. SOLO is low in expressiveness, though it is highly usable. The aim behind the Hank project was to develop a language which found a more appropriate trade-off between expressiveness and usability for our expected users, drawing on recent work in fields such as end-user programming, software visualization and programming by demonstration. The language had to be expressive enough to model the kinds of theories the students met within the cognitive psychology course, while still being usable by the non-programmer. An unusual requirement was that the language had to be capable of being used on paper, as well as through a computer implementation.

This is because the students' first attempts at cognitive modelling are carried out individually, at home, usually without recourse to a computer.

The rest of this paper is structured as follows. The next section outlines the design requirements which motivated the language, and reviews the literature related to each of these requirements. This is followed by a description of the language itself, evaluation findings and plans for future work. The conclusions draw together the lessons we have learned from the design and use of Hank.

2. Design requirements and issues

A new language design should be firmly grounded in a profile of the intended user population [8], identifying their programming needs and capacities. The requirements that motivated the design of the Hank language can be divided into five areas. First, the language needed to be appropriate for psychology students wishing to build cognitive models. Second, it had to consider the requirements of the non-programmer, a group within which most cognitive psychology students fall. Third, the language had to be suitable for students working in groups. Fourth, the language had to be good at showing the execution path, as well as the output of any program. Fifth, the language had to be usable on paper as well as on a computer. It is worth noting that some issues were deliberately ignored during the design process: whether or to what extent the language was Turing complete; and the ease with which the design could be implemented. Equally, no initial assumptions were made as to the inherent value of particular programming paradigms or visual formalisms.

The requirements identified, the related findings, and their relation to the design decisions within Hank are summarised in figure 2.

2.1. A cognitive modelling language for psychology students

Hank is intended primarily to be a cognitive modelling language for psychology students. For this reason it has to bear a clear relation to the cognitive theories and architectures found within the

cognitive psychology course studied by the students. The overall architecture of the language must therefore be related to, and help to further inform the students of key concepts within the area of cognitive psychology. The overriding notion the architecture must relate to is the information processing framework which underpins much of cognitive psychology. It must also help to inform students of how major cognitive architectures from an artificial intelligence background such as ACT* [9] and SOAR [10], influence developments in cognitive psychology.

The last attempt to develop a usable language for the course was the SOLO language [5]. SOLO was suited to building a limited number of models. SOLO was a textual language which represented links between individual concepts (see figure 1) which when taken together formed a semantic network. SOLO was usable by psychology students, though it was particularly limited by its restricted semantic network structure, which was less appropriate for understanding and modelling developments in cognitive modelling and cognitive psychology taking place during the 1980's.

[Insert figure 1 about here]

This led to the eventual demise of SOLO. It was replaced by the Prolog programming language. Prolog is a far more powerful programming language, though is difficult to use, particularly to the psychology student with no previous programming experience, and perhaps even no previous experience of computers (though this has become much rarer over the past few years). A particular problem of Prolog is the inaccessibility of its control flow model, despite attempts to provide tools to help the students gain a better understanding of it [11]. This creates a problem as the modeller needs to understand how the model works in order to reflect on the ramifications of their model for the theory it represents. Others have attempted to make the Prolog programming language accessible to students by focusing on giving the students a flavour for the language, rather than expecting them to program for themselves (e.g. [2, 3]), though it is unclear what students actually learn from this experience. The Hank language aims to learn from the difficulties of Prolog and the previous success of SOLO to develop a contemporary cognitive modelling language for psychology students.

2.2. Usable by non-programmers

The Hank language has to be usable by non-programmers, as the majority of the students have no previous programming experience. Many systems have demonstrated the advantage of using a grid-based layout to help non-programmers lay out, design, run and understand their programs. These include spreadsheet applications, which are probably the most widely used end-user programming language. Agentsheets used a grid-based layout and graphical rewrite rules to open up programming to children, as well as being used as a substrate for numerous domain-oriented applications [6, 7, 12]. Similarly, CoCoa (formerly KidSim) [13] uses a grid and graphical rewrite rules to help children write their own programs.

[Insert figure 2 about here]

One problem students have with Prolog is dealing with the unforgiving syntax. For example, within Prolog, a space before an open bracket results in an error, even though this is expected in English. Although such trivial bugs are unlikely to phase the expert, they can cause problems for the non-programmer. Hank draws on work using grid layouts to provide a usable alternative to many of the syntactic constructs common in textual cognitive modelling programming languages. It should be noted, though, that grid layouts do not "do away" with syntax, though they do seem to make the syntax more accessible.

2.3. Usable in groups

Another requirement is that the language should be usable in groups. The students are required to build a cognitive model as part of a residential school project. At the Open University, students attend a residential school for one week during the summer and work on projects in groups of four or five. It has frequently been noted that group work is difficult with a programming language such as Prolog, and there is a tendency for a division of labour to occur where certain students will take the lead (and the keyboard), whilst others may drop behind and benefit less from the project. This appears to

happen for two interrelated reasons. First, the textual Prolog notation does not lend itself to group work, making collaboration difficult. Second, as Prolog is difficult to use, those with computing experience take on the programming duties.

Nardi and Miller [14] found that spreadsheets tend to be good for group work, allowing a number of interested parties to understand how the spreadsheet is being developed and see what the spreadsheet is intended to achieve and how. This claim has been tempered by the results of Hendry and Green [15] who found that certain aspects of the spreadsheet can be difficult to understand, in particular its procedural aspects. There is, though, some evidence that spreadsheet style layouts are better for group work than conventional textual languages.

2.4. Showing the process

Another important requirement is that the language should be able to show its procedural aspects. The current language, Prolog, is designed to focus on the declarative aspects of programming, and essentially hides the procedural aspects. A central goal of teaching cognitive modelling to psychology students is to show how different models, though producing similar results, may work in very different ways, some being less psychologically plausible than others. To use Gilmore's [16] terminology, the new language must be good at showing "how it works" as well as "what it does". Prolog, then, often hides what a good cognitive modelling language should reveal.

Spreadsheets, while very popular, are only good at presenting a declarative view (i.e. what it does) and hide the procedural view. Hendry and Green [15] and Saariluoma and Sajaniemi [17] found that spreadsheet users have difficulty mapping between the surface structure and the underlying (hidden) computational structure. Hendry and Green [18] developed the CogMap tool to help users make this mapping. A spreadsheet structure, while having many advantages, cannot be relied upon to present an adequate procedural view. The converse problem was found with the flowchart notation, which was intended to help programmers understand code. Flowcharts were found to be good at showing control flow but at the expense of data flow [19-21].

Software visualization has also been heralded as a useful way of showing how a program works, though Mulholland [22, 23] shows that for software visualization to be effective at showing process, it has to provide adequate support for the user. In particular it has to help the user map, review and test: map effectively between the visualization and code; review previous events in the execution from any particular point; and be able to predict and test for future events within the execution.

Also, within the field of Programming by Demonstration, comic strip notations have been developed to help people view and understand a process built automatically from user actions (e.g. [24-27]). A particular advantage of the comic strip notations is that they show how specific events in the execution fit together by representing them as a set of before and after states. However, Traynor and Williams [27] have found that although comic strips are very good at showing a linear process, they have problems representing more complex control flow constructs such as branching and looping.

2.5. Usable on paper

As our students cannot be required to have a computer, a major requirement of the new language is that it must be usable on paper, as well as on a computer. Little work has been done on how people are able to write and run programs on paper. Whalley [28] observed children making and running simulation programs on paper and found they had a strong tendency to get the result they wanted regardless of whether the program was correctly specified. There is no age bar on this behaviour, even when not programming on paper. It is common for programmers to draw inappropriately on real world knowledge, and expectations of what the program should do (e.g. [29]). Whalley [28] found the only way to show the children the disparity between their expectations and the program itself was to enter the program into the computer and watch it work. Showing a program working on paper is a difficult problem. A valiant attempt was made by Ford [2] who provided sliding cards to show how particular Prolog programs worked, though this was restricted to canned examples, and could not be used to help a student run a program of their own making on paper.

Some work which may help in devising a method by which students can run a program on paper without too much interaction from their own expectations is research on syntonicity [30, 31]. Papert [30], in the development of LOGO provided a (non-anthropomorphic) turtle that the students could identify with, in order to help them "get inside" their programs. In the case of Hank, we intend to help students identify with the program, so that they may be able to leave behind their assumptions as to how the program *should* work, and instead run the program as specified.

3. The Hank language

The design of the Hank language drew on the requirements and issues outlined in the previous section (see figure 2 for a summary). This section presents five main parts of the Hank language, starting with the overall architecture presented to the students. The two main programming constructs are then presented: fact cards and instruction cards. This is followed by a description of the storyboards which are used to visualize the execution of Hank programs. Finally, the executive component is explained, which contains the story of how the Hank programs are run. Hank runs using a reversible interpreter. Hank is implemented in Allegro Common Lisp and runs on Windows PCs.

3.1. Architecture

The architecture of Hank is a conventional information processing architecture, familiar to cognitive psychology students (see figure 3). It has three components: a place where information can be stored and represented (called the database); a place where information can be worked upon by processes (called the workspace); and an executive component which is responsible for carrying out any processing and also dealing with input and output (called the executive). The question processor works according to a set of house rules that describe the execution model.

[Insert figure 3 about here]

3.2. Fact cards

Fact cards are what we use in Hank to represent factual information (see figure 4). These adopt a familiar table structure, used in spreadsheets. The first row is the name of the table (shown in dark grey). The second row gives a label for each individual column (shown in light grey). The remaining rows (called data rows) each represent a related set of symbols. Two fact cards are shown in figure 4. The fact card on the left describes which people are liked, and by whom. For example, the first data row represents that John like Pavarotti, and the second data row represents that Susan likes Bob Dylan. The fact on the right can be read as Pavarotti sings opera and Bob Dylan sings folk. Fact cards can have any number of columns and any number of data rows

[Insert figure 4 about here]

3.3. Instruction cards

Hank also provides instruction cards (see figure 5). An instruction card simply represents a procedure that can be used to work something out. The top part of the instruction card, above the double horizontal line, is called the matching box. The matching box defines the goal of the instruction card. In this case the goal is to find out whether a person is cultured. The instruction card defines a person as being cultured if they like someone who sings opera. The bulk of the instruction card, below the double line, is called the process box. The process by which the goal can be achieved is defined in this box.

[Insert figure 5 about here]

In Hank, variables are termed wildcards. They start and end with a question mark. The matching box contains the wildcard ?Who?. This will match against the person fitting the definition of being cultured. The process by which someone is defined as being cultured is represented in the process box as a set of questions. The process begins from the top left hand corner of the box. The first step finds

out who the person likes. The ?Who? variable as well as appearing in the matching box, also appears in the first column of the likes question in the process box. This means that the cultured person is the one who does the liking. The person who is liked is stored in another wildcard called ?Other?. The 'Likes' question is joined to the 'Sings' question by an OK arrow. If an answer is successfully found for the 'Likes' question then the next question can be tried. Other kinds of arrow can be used. For example, a Fail arrow is followed is an answer to the question cannot be found.

The second question determines whether the person who is liked does sing opera. The ?Other? wildcard is used in the first column to indicate that the singer must be the person who is liked, determined in the first question. The second column contains a constant rather than a variable, as the person must be an opera singer. From a design viewpoint, instruction cards combine the spreadsheet structure of the fact cards with a flowchart notation to indicate flow of control. Hank supports both iterative and recursive control flow constructs.

3.4. Storyboards

Storyboards are a form of comic strip notation used to indicate the serial order of processes and show their causal relationship. This is the visualization mechanism within Hank. The storyboard representation of the execution is shown in the Workspace window (see figure 6). The storyboard below visualises the execution for the instruction card presented in the previous section.

The storyboard unfolds both sideways and downwards, providing a scalable overall perspective on the execution. Using the storyboard, the student can run through the program until something unexpected happens. The student can then double-click on the storyboard cell to go to the related part of the program. The design of the Hank storyboard notation combines work in comic strips from Programming by Demonstration with findings from the evaluation of Software Visualizations. Storyboards do not suffer from Traynor and Williams' [27] finding that it is difficult to represent complex control flow constructs using comic strips. Storyboards are only used to show the execution, which is linear. The program itself is defined using instruction cards, which do support control flow constructs.

[Insert figure 6 about here]

3.5. The executive

The executive is the final part of the Hank architecture. The executive makes Hank work. It is responsible for running any process, and dealing with input and output. It has two main components: the 'house rules' which describe how the language should work, and Fido, the worker responsible for carrying out the processing, using the house rules. The house rules dictate how input should be handled, how and when output should be sent, when information should be sought from the database, and what should be written in the workspace. Fido (a dog), is the worker inside the architecture that makes things happen by following the house rules. In the computer version of Hank, Fido's job is done on the students' behalf and visualised in the storyboard. In the computer version of Hank, questions are sent to the executive using the control panel (shown in figure 7). There are options to ask the question either briefly or in full. If the question is asked briefly, then the answer is displayed in the lower part of the control panel. If the question is asked in full, then the execution is visualized in the workspace, using one or more storyboards.

[Insert figure 7 about here]

When using Hank on paper, the student runs the program themselves by following the house rules. Blank storyboards are provided to aid the student in coming to an answer by precisely working through the execution one step at a time, rather than drawing on real world knowledge to estimate an answer. The design of the executive and its components draws on research in programming on paper and syntonicity to provide an executive component the user can identify with.

4. Scenario: Building a model in Hank

In this section, we will show a more substantial example of a model built in Hank to show that it is scaleable to significantly larger models than we have shown so far. The example we will use is based on the 'Towers of Hanoi' puzzle problem discussed by Simon [32]. This puzzle, shown in figure 8, involves moving a number of rings from one peg to another, with a constraint that a ring can only be placed either direct on the base or on a larger ring.

[Insert figure 8 about here]

Simon's analysis of this puzzle is quite typical of larger cognitive models, in that he uses it to compare different theories of (in this case) human problem-solving behaviour. He used the GPS (General Problem Solver) production system to build models of some of the different strategies that a human solving the puzzle might use — and then these models can be compared to actual human performance when solving the puzzle.

To use Hank in the same way, it must be possible to build models of the different strategies, and to compare these models with human performance data and evidence in a sensible way. However, there is more to modelling than this; it is also very important that the models are comprehensible to psychologists. In this scenario, therefore, we will make the model as close to Simon's GPS model as possible — so that the correspondence between the two is clear. In practice, this means building a small production rule system in Hank.

But before we get to any of these issues, we need to represent the state of the puzzle. This is the same whatever particular strategy we intend to model. In Hank, things that are simply known, like the current state of the puzzle, are represented using fact cards. For example, the fact card that represents the state of the puzzle in figure 8 is shown in figure 9. In this case, each data row of the fact card relates a ring in the puzzle to the peg it is currently on and the object it is resting on.

[Insert figure 9 about here]

Now we can begin to model the processes that a puzzle solver might use to interact with the puzzle world. First of all, let's look at the particular strategy that we're going to model. This is the strategy Simon called the 'Goal-Recursion Strategy'. Although it is not the simplest possible strategy, it is one of the more plausible strategies for a human puzzle solver, because it doesn't require them to remember many intermediate goals. There is a common variant on the goal recursion strategy, also discussed by Simon, which is rather simpler for a computer, but which requires a perfect memory for intermediate goals — something that people do not have. The model described in this section is more complex, because it models a human's use of perception to make up for a limited short term memory. This strategy, as Simon originally presented it, is shown in figure 10.

[Insert figure 10 about here]

Hank isn't quite this concise, but a production rule model like Simon's deliberately leaves out any representation of the flow of control between these rules, as well as a host of auxiliary functions. A screen snapshot showing the main production system, and one of the rules, is shown in figure 11.

[Insert figure 11 about here]

The process bits of the model, the bits that actually implement the model's strategy for changing the puzzle world, are represented by instruction cards in Hank. Figure 11 shows two instruction cards, a top level one called 'Towers of Hanoi', and an instruction card representing Simon's rule P3, called 'Rule P3'.

Hank's execution model is simple. Every question is put to the executive, which then answers it and gives it a status value which is 'OK' if the question succeeded, and 'Fail' if it didn't. When a question is asked against a fact card, the answer given is the first matching row in the fact card — and if no rows match the status is 'Fail'. When a question is asked against an instruction card, the questions in the body of the instruction card are then asked one at a time, and after each one a link is followed

which matches that question's status. If there isn't a matching link, the instruction card stops at that point.

The instruction card 'Towers of Hanoi' models a small production rule system. This instruction card is a loop; each time round the loop the instruction card first checks to see if the puzzle has been solved — if it has then the instruction card stops. Otherwise, the 'Fail' link is followed to the built in question 'Ask All'. 'Ask All' questions are handled specially; each time an 'Ask All' question is asked you get the next match from a fact card, until there are no more matches, when the question fails. In this case, the fact card is a list of rule names, so this runs as a loop, trying instruction cards corresponding to the rules P2 to P6, until one of them succeeds. Then you go back to the beginning of the instruction card and start again.

The other example instruction card, 'Rule P3', is very different in structure. This directly corresponds to one of Simon's production rules. The rule instruction cards do not loop, they just test a condition, and if it is OK they do a set of actions and end with an OK status. All the production rules use a convention: the condition 'if' part is on the left hand side of the process box, and the conclusion 'then' part on the right. All the rule instruction cards look pretty similar, therefore, with a chain of conditions linked by 'OK' links on the left, the last linking to a chain of actions linked by 'OK' links on the right.

In an instruction card, variables can hold values to be passed from one question to another, and into and out of the instruction card. Looking at 'Rule P3', for example, the second question in the instruction card will try to find values for the variables '?Ring?' and '?To?', but if there is no matching goal in short term memory (represented by the fact card 'STM') this part of the rule's condition will fail. If the condition succeeds, however, ?Ring? and ?To? will now hold values that can be used in the action part of the rule — in this case, the instruction card 'Move' will be called to move the ring in ?Ring? to the peg ?To?.

Significantly, Hank's fact cards directly reflect the state of the puzzle. The procedure for moving a ring, for example, works by removing that ring's row from the 'State' fact card, then adding a new

one with different ‘Peg’ and ‘Upon’ column values. As the fact card is changed (by pre-programmed questions that Hank’s executive knows how to interpret specially) the fact card changes immediately on the screen. In this model, pre-programmed questions which change fact cards, such as ‘Push’ and ‘Pop’, are also used to represent the strategy’s use of short term memory, and the ‘STM’ fact card also changes as the puzzle is solved, showing how the contents of short term memory changes over time.

When the model is complete, the computer implementation allows you to run it by passing a question to the executive, and (if you want) to see the full behaviour of the executive. Questions are sent to the executive using the control panel, which also keeps a ‘transcript’ record of previous questions and answers. Questions can be asked in two ways, briefly, to simply run the model (watching the fact cards change as the model operates) and in full, which will display a step by step account of the behaviour of the executive, using a storyboard notation in the workspace window. Figure 12 shows the Towers of Hanoi model running, with the workspace window and the control panel both open.

[Insert figure 12 about here]

Part of the storyboard representation of the execution of this model is shown in the Workspace window to the bottom of figure 12. In this example, only a few rows are shown, indicating where the ‘Towers of Hanoi’ production rule interpreter is up to in its list of rules, and a few production rules that have been tried unsuccessfully.

In the computer implementation, Hank’s storyboard notation shows each call of an instruction card as a new row, with the questions within it growing horizontally and linked by small boxes showing the status after each question. Also, within the workspace, questions are shown first in red (at the ‘ask’ step), and then in black (at the ‘answer’ step), as the question is completed and Hank prepares to move on to the next question. Each production rule, therefore, shows up as on a row by itself, showing the condition questions, followed (if the condition is satisfied) by the action questions.

Finally, as we mentioned briefly earlier, it is important to be able to compare the behaviour of a model like this, with experimental data from studies of human puzzle solvers. To help with this, Hank shows a step count in the workspace. Although no literal comparison is meaningful between the time taken by a human, and the number of steps taken by a Hank model, relative comparisons are useful. Some moves take longer than others do. In practice, the pattern of move times modelled for this strategy does not match human experimental data at all well. But this is the whole point behind modelling: it allows theories to be accurately compared with experimental data, and when the two don't match, this just means that, as with this strategy, the theory needs to be revised.

5. Using Hank in a paper and pencil assignment

An initial study into Hank was undertaken using a draft version of the paper and pencil assignment. The assignment provides an introduction to cognitive modelling and covers issues related to knowledge representation, category membership and schema theory. The assignment is split into two parts. The first part requires the student to build and run some simple cognitive models. The second part involves writing an essay covering the wider issues surrounding cognitive modelling and artificial intelligence in general. For the study only the first part of the assignment had to be carried out. The programming part of the assignment was sent out to three participants, a student who had previously studied cognitive psychology (including modelling using Prolog), a student who was about to embark on the cognitive psychology course, and an experienced course tutor. The three participants were required to complete the modelling exercises plus a short questionnaire and then return their completed assignment. The exercise was performed under similar conditions to the paper-based assignment. The participants worked on the exercise at home, following a draft version of the instructions that were incorporated into the final assignment guidelines.

Each participant was able to build small programs using fact cards and instruction cards. It was particularly pleasing that the students were able to use the storyboards to run their program, though there was some evidence of students getting lost in the storyboard, and not gleaning what they might have done, with the larger execution histories. Despite this, the study did provide some evidence that

two of our design objectives had been addressed, since Hank could be used with success on paper alone, without support from a computer implementation, and by non-programmers.

This exercise was intended to be little more than a sanity check on our part to make sure that the psychological concepts, as portrayed in Hank, were accessible and usable by the students. In this respect, perhaps most significantly, the student with no previous experience of cognitive modelling commented that the experience had given her a much more positive attitude to this area of the course, which she would meet later in her studies. This encouraged us to carry out a more detailed study of using Hank at a residential school.

6. Using Hank at residential school

At a week long residential school, students get the opportunity to carry out a cognitive modelling project for two and a half days in groups of between three and five. The project involves building a rather more complex model than those introduced in the assignment, this time with the help of a computer. For two of the residential school weeks this summer, students were given the opportunity to build their cognitive models using Hank rather than Prolog. Four groups took up the challenge. The authors took responsibility for tutoring the Hank groups.

For each of the participating groups, the cognitive modelling project began at 9am on a Wednesday morning. The process of introducing the language (which involved unlearning certain Prolog concepts) and teaching them how to use the computer implementation of Hank took until lunchtime. The students began their projects in the afternoon session. Groups using Prolog, because they already have some familiarity with the language, usually start their project around 11am. The Hank groups had caught up with their Prolog counterparts by mid morning on Thursday, and had a working model by the end of the project. The groups were able to demonstrate their understanding of the model by explaining to the tutor how it worked, and what it meant psychologically. At the end of the project, each group was interviewed separately. The semi-structured interview comprised eight questions. These are shown below:

- Did the Hank language help you to understand any elements of cognitive psychology? If so which?
- What things did you find easy or straightforward to do using Hank?
- What things did you find hard to do using Hank?
- Can you suggest any improvements to the Hank language?
- What are the differences between Hank and Prolog?
- Have you any ideas on how Hank should be explained to someone who has never seen it before?
- For AI this year, would you have rather used Hank or Prolog?
- In your opinion, should students next year be taught using Hank or Prolog?

Their responses were transcribed by the experimenter. Below, eight key points from the feedback are identified. Their comparative comments of Prolog are not only based on their experience in using Prolog in an assignment earlier in the year, but also on their discussions with Prolog students. One group even invited a number of Prolog students into their room to give them a demonstration of Hank.

1) The removal of the textual syntax proved very successful

A typical comment was:

> "It was much better not having any commas, full stops and brackets"

Although remembering to place these Prolog syntactic constructs in the right place is trivial to someone experienced with the language, it can cause problems when the students are not particularly familiar with the language, and have a number of other issues to consider. One student made an analogy between Prolog and using DOS rather than Windows to move files around. Removing the textual syntax had a huge impact on the students, regardless of their experience with computers. On one extreme, there was a student who had never used a computer before, who was soon able to get the hang of constructing cards and running the program. On the other hand, there were some students who

used spreadsheets extensively as part of their job, and felt immediately comfortable with the notation and how it worked.

2) The students were able to relate their Hank programming to the course materials

Students were able to draw appropriate links between their Hank project and the course materials. This was found in comments such as:

> "Hank emphasises the model not the computer programming"

In particular, the link to schema theory was very clear. This was illustrated in comments such as:

> "It has a direct relation to schema theory, far more so than Prolog, due to the column names"
> "It distinguished between specific and general information in schemas"

The tabular structure of facts in Hank is very close to the way schemas tend to be represented in cognitive psychology textbooks, as a set of slots and values. This is illustrated in figure 13.

[Insert figure 13 about here]

3) Students were able to understand each other's programs

As the students could identify how their programs related to psychological theory, they were more able to see connections not only between their program and the course texts, but also between their program and other Hank programs. This was illustrated in comments such as:

> "I could see the analogy between the blocks world program [their project] and the families program [a warm-up program tried out at the beginning of the project]"
> "You could see that they were similar on a deeper level"

"I can see how we could do other models using Hank, such as language understanding"

The students were able to understand other programs because the programs were no longer being compared in terms of syntactic similarities, but in terms of the theories that were being modelled. The students were even able to see how Hank could be used in domains other than cognitive modelling. This is nicely illustrated by a comment from one student who worked as a gardener.

"Hank could be used in real life. As a gardener I could use it to keep a database on trees: level of shade and light, growth rate. I could write a rule to find the best conditions, a bit like our cousins program"

This comment is particularly encouraging as the student thinks not only about how Hank can be applied in another domain, but also how the Hank program would have similarities to another program (called the 'cousins' program) that the group had developed when familiarising themselves with Hank on Wednesday morning.

4) Students could understand the process as well as the result, though syntonicity did not work as expected

Prolog has a very complex execution model, therefore even if the students' program is working correctly, they may not know why. In Hank, the students could clearly see the connection between their program and how it worked. This is illustrated in comments such as:

"I liked the "Ask fully" option [used for generating storyboards]. It was easy to see where we had gone wrong"

"The process of getting to the answer can be clearly seen"

An aim in our design was to use syntonicity to help students identify with the process. It had been our intention that students should identify with Fido, the interpreter, but this did not occur. The word

'Hank' was used as a concise way of referring to their model and its behaviour, but this could reasonably be interpreted as identification with a character.

> "It's good being able to work through the program but I'm not sure about Fido though"
>
> "We liked to personalise Hank, by saying Hank does this, Hank does that"

We will return to these issues later in the paper.

5) Their approach was confident and exploratory rather than nervous and tentative

When Prolog students reach an impasse it is very difficult for them to explore the language in an attempt solve the problem without the tutor's help. Hank students were more keen to explore and try different things out when their model was not working correctly.

> "You are more willing and able to explore in Hank"
>
> "Basically, Hank is fun. You can fiddle with it and see what it does"

6) The learning curve was more linear and determined by theoretical rather than programming concerns

Because of the more exploratory nature of the language, the learning curve was more linear. This contrasts with Prolog which tends to comprise one or two large jumps.

> "You can test and change you program. Prolog either works or messes up"
>
> "Hank appears intuitively easy and then gets hard later on. With Prolog, you start thinking it's impossible, and then it gets harder"
>
> "Hank goes from easy to hard, and then to easy. Prolog looks hard"

7) Some minor interface issues need to be resolved

Some interface issues were raised by students, concerned with the way objects are selected and moved.

> "The distinction between normal questions, and questions inside the rule is unclear"
>
> "Sizing boxes and columns was sometimes difficult"
>
> "The scrolling was quite bad. It was too slow"

These issues have already been rectified in the current version of Hank.

8) Working with Hank was enjoyable

Learning with Hank was enjoyable as well as effective.

> "Hank is pleasing to look at"
>
> "Hank is fun"

From a tutor's perspective, a number of issues were raised concerned in particular with the relationship between the Hank language and the educational experience as a whole.

1) Hank offers many alternative solutions to the same problem, and many routes to them

A major difference between Hank and Prolog is that in Hank, the same problem can be solved with equal effort in a number of different ways. This requires a lot of thinking ahead on the part of the tutor, who has to discern which solution the students seem to heading toward, and how they should be guided there. This illustrated the need to rethink the relative pedagogical benefits of alternative paths. We are currently using the educational walkthrough technique [33] to chart how the programming process undertaken by the students links to educational objectives. Although the proliferation of

solution paths placed extra demands on the tutor guiding the students' progress, this is a welcome and positive outcome. Different groups of students modelling the same theory could discuss the difference between their models, and their relevance in terms of psychological theory.

2) Time devoted to meta-talk about the design itself and how it compared to Prolog

The students were not treated as naive subjects in an experiment. We often got into discussions with the students on why Hank was designed in such a way, and how that related to concepts they had studied in their psychology course. For example, a discussion took place comparing programming in Prolog and Hank, in the way that homomorphic problems are compared in their course text on human problem solving [34]. Another student with a knowledge of HCI suggested how Hank could be understood using the notion of affordances as described by Norman [35]. This conscious decision to encourage the students to be joint participants in a design process [36], rather than passive subjects, led to many interesting suggestions and observations.

3) Motivation, enjoyment and teamwork

The students were highly motivated, exploratory, and keen to try things for themselves. Hank became a tool for thinking with (which is what cognitive modelling languages should be) rather than a pedantic machine holding them back. As commented by the students, it was far easier to understand Hank programs written by other people, compared to Prolog. This led to some rather complex group-working arrangements, where different members of a group would work on different parts of the program, or different solutions to the same problem, before coming back together to compare their ideas. We were amazed by how smoothly this approach worked.

Overall, using Hank at residential school in place of Prolog was found to provide a very successful educational experience. Not only were the students able to complete the project (a feat in its own right given they were seeing the language for the first time), they had also clearly learned from the

experience. They were also able to draw strong links between the Hank project and key issues and theories from within the course.

7. Changes to Hank

The main changes taken in light of the work so far were downplaying the role of Fido, simplifying the structure of the house rules, and some terminological changes. All of these changes were incorporated into the final version of the course module.

Shift in focus from Fido to Hank

Our original notion of how to use syntonicity [30, 31] to encourage identification with (leading to an understanding of) the program did not work as expected. The students were better able to understand the process, but did not identify with Fido sitting inside.

Originally, Fido had been intended to let the students know that there was nothing magical about the executive, that (like a computer) it was simply following a precise set of rules, in this case, the house rules. Now that computers are becoming ever more commonplace, showing people that there is nothing mystical about how the computer works is becoming less important, and therefore there is less need to illustrate this point.

Students often made syntonic sounding comments at the level of Hank, of the kind "Hank is trying to do this". In most cases this was a concise method for referring to their model rather than evidence of syntonicity. For this reason, we have since decided to play down the role of the interpreter in the execution model we present to the students. As a result, the Hank architecture has been reframed (see figure 14). The term 'question processor' is now used to encompass the whole executive, including the role previously taken by Fido. Instead of supporting the comprehension of the execution model by encouraging identification with the query processor, the description of the execution story as presented in the house rules was made simpler, but scaleable.

[Insert figure 14 about here]

Simple but scaleable house rules

So far, the house rules had been presented as a fine-grained recipe of how programs should be run. Although the students needed to be given a detailed story of how the programs are run, these house rules resulted in students getting bogged down in detail, losing an overall perspective on the execution of the model. In our revised version, the presentation of the house rules has been simplified to just a very small number of bullet points, each with a supporting paragraph of what work that bullet point involves. In the new approach we are designing for growth in competence. As the students become more familiar with Hank, the details pertaining to each bullet point become operationalized [37]. In the original house rules, there was no scope for students to operationalize parts of the process as their familiarity increased. It is worth noting that these simplified house rules seemed to converge on a two step, ask - answer cycle. This both simplified the execution model, and related it more clearly to the architecture in figure 14.

Changes to terminology

A few minor changes in terminology have also occurred. Wildcards have been renamed as variables. This is to remove any possible confusion with Fact Cards and Instruction Cards. In order to help students in designing an Instruction Card, the top part of the instruction card is now named the "wish box" rather than the "matching box". This terminological change developed out of discussions with students at residential school, where they designed their instruction cards, by first typing into the top part of the instruction card what it was they wanted to find out, and then moving on to work out how it could be produced.

8. Further work

So far, Hank has been used to allow students to gain a useful first experience of cognitive modelling. We now wish to develop Hank further to investigate how psychologists can be supported in developing more complex models. Some current and planned developments to Hank involve extending support for cognitive architectural features [9, 10] concerned with issues such as capacity and latency in memory. Other new features are to support book-keeping and the initialisation of models, such as mathematical functions. These developments, though, are not just restricted to adding more primitives to the existing Hank design. We also have plans to build a model level layer where components of the Hank program can be tied to particular modules within the box-and-arrow style models prevalent in cognitive psychology.

The further development of Hank is linked to an ongoing programme of evaluation, and of developing real and substantial cognitive models both informally and through more formal walkthroughs [33], to ensure that the language can handle models of a rather larger scale than those met in the studies we have described. This includes an independent summative evaluation of Hank, being conducted independently of the design team. This development programme is constrained within the existing Hank language framework, so that its benefits are maintained. This is giving us a good understanding of which changes to make, and how to make them safely.

9. Conclusions

Our initial analysis has largely supported our initial design of Hank in enabling students to develop cognitive models on paper and on the computer, and to use those models to illuminate their understanding of cognitive psychology. Let's look at this in a bit more detail, returning to our original design objectives, set out in section 2. A lesson learned has been attached to each objective. We are not claiming that all of these lessons are novel, but we hope that collecting them together, based on our own experiences as described above, will help designers of other end-user and educational modelling languages.

- *A cognitive modelling language for psychology students*. Both studies showed that Hank helped to cut through to the psychological concepts of the model being built, bypassing messy programming. Hank's domain-orientedness seemed to be a key strength for this, as the parallel between schema and fact card notations in figure 13 illustrated. However, perhaps Hank's most important characteristic was that the design encompassed more than the visual language itself. We introduced a terminology and a story along with the notation.

  Lesson 1. Help the end-user by appealing to their domain, through not just the notation of the language, but also the execution story, terminology and supporting materials.

- *Usable by non-programmers*. Again, both studies provided evidence that non-programmers could use Hank, and at the residential school, the computer implementation was even used successfully by students who had never used a computer before. In large measure, this was due to the use of grids, flowcharts, and the speed with which changes could be tried out. All of these tended to reduce the role that computer skills and previous programming experience played in the modelling process.

  Lesson 2. Visual language notations such as grids and flowcharts can successfully lower the syntactic burden and allow the end-user programmer to concentrate on the real task.

- *Usable in groups*. Groups worked well in the residential school study. An unexpected change was that because Hank programs were generally easier for people to understand, the collaborative processes were very different to those found with Prolog groups. We found that the most important contributory factors to Hank's success in this area were its use of grids and flowcharts. The transparency of reading that these provided and the reduced need for special programming skills allowed Hank groups to use more complex group structures than the Prolog groups' typical 'all for one, one for all' format.

Lesson 3. End-user, grid-based languages can successfully support work in groups by lowering the barrier to participation and providing an environment in which ideas can more easily be represented and shared.

- *Show the process*. Both studies provided some evidence that the process was more apparent. Although students found large storyboards on paper difficult, with the computer implementation, there was clear evidence that they did begin to understand how their models had worked more fully than they had with Prolog, which tended to work by 'magic'. Storyboards, along with the intuitive control flow and the simplified house rules, seemed to make it much easier for students to understand the behaviour of their Hank models.

Lesson 4. The storyboard notation from the Programming By Demonstration community and simple but scaleable execution stories can be used to described relatively complex processes to end-users.

- *Usable on paper*. The home-based, paper and pencil study, confirmed that Hank could be used on paper, at least to a limited degree, and without a tutor being present in person. This is largely due to Hank's use of grids, flowcharts, and storyboards, which worked so well they could be used on paper. We actually found that the constraint of supporting modelling on paper assisted in the design process, and made for a better language than might have been possible otherwise. The requirement of the paper-based version forced us as designers to devise a language that supported people in "walking through" their model to understand how it worked, which also benefited the computer-based version.

Lesson 5. Designing for use on paper can improve a computer based, end-user programming language by encouraging the designers to support the need to walk through a program in order to understand how it works.

Perhaps most surprising, though, has been the sheer engagement that Hank managed to bring to the students at the residential school. On the whole, they simply revelled in it, and many found it "fun" compared to Prolog. Perhaps the motivational benefits of this engagement are also playing an important role in the students' successful work with Hank. As a project, Hank is still proving a rich source of useful research for the future.

References

1. M. Eysenck & M. Keane (1995) *Cognitive Psychology: A Students' Handbook* Lawrence Erlbaum Associates, Hove, East Sussex, UK.
2. N. Ford (1987) *How Machines Think: A General Introduction to Artificial Intelligence Illustrated in Prolog* Wiley, Chichester.
3. P. Scott & R. Nicholson (1991) *Cognitive Science Projects in Prolog* Lawrence Erlbaum Associates, Hove, Sussex, UK.
4. M. Eisenstadt (1978) *Artificial Intelligence Project: The SOLO Primer* The Open University Press, Milton Keynes.
5. M. Eisenstadt (1983) A User Friendly Software Environment for the Novice Programmer. *Communications of the ACM* **26**, 1058-1064.
6. P. Fung, M. Brayshaw, B. du Boulay and M. Elsom-Cook (1990) Towards a taxonomy of novices' misconceptions of the Prolog interpreter. *Instructional Science* **19**, 311-336.
7. A. Repenning & W. Citrin (1993) Agentsheets: Applying Grid-based Spatial Reasoning to Human-Computer Interaction. Presented at IEEE Workshop on Visual Languages, VL '93.
8. A. Repenning & A. Ioannidou (1997) Behavior Processors: Layers Between End-Users and Java Virtual Machines. Presented at IEEE Symposium on Visual Languages, VL '97, Capri, Italy.
9. J. R. Anderson (1983) *The Architecture of Cognition* Harvard University Press.

10. A. Newell (1990) *Unified Theories of Cognition* Harvard University Press, Cambridge, Massachusetts.

11. P. Mulholland & M. Eisenstadt (1998) Using Software to Teach Computer Programming: Past, Present and Future. In: *Software Visualization: Programming as a Multi-Media Experience* (J. Stasko, J. Domingue, M. Brown, and B. Price, eds.) MIT Press, Cambridge, MA.

12. A. Repenning & T. Sumner (1995) Agentsheets: A Medium for Creating Domain-Oriented Visual Languages. *IEEE Computer,* March 1995.

13. D. C. Smith, A. Cypher, & J. Spohrer (1994) KIDSIM: Programming Agents Without a Programming Language. *Communications of the ACM* **37**, 55-67.

14. B. A. Nardi & J. R. Miller (1991) Twinkling Lights and Nested Loops: Distributed Problem Solving and Spreadsheet Development. *International Journal of Man-Machine Studies* **34**, 161-184.

15. D. G. Hendry & T. R. G. Green (1994) Creating, Comprehending and Explaining Spreadsheets: A Cognitive Interpretation of What Discretionary Users Think of the Spreadsheet Model. *International Journal of Human-Computer Studies* **40**, 1033-1065.

16. D. J. Gilmore (1991) Models of Debugging. *Acta Psychologica* **78**, 151-172.

17. P. Saariluoma & J. Sajaniemi (1994) Transforming Verbal Descriptions into Mathematical Formulas in Spreadsheet Calculation. *International Journal of Human-Computer Studies* **41**, 915-948.

18. D. G. Hendry & T. R. G. Green (1993) CogMap: A Visual Description Language for Spreadsheets. *journal of Visual Languages and Computing* **4**, 35-54.

19. N. Cunniff & R. P. Taylor (1987) Graphical vs. Textual Representation: An Empirical Study of Novices' Program Comprehension. In: *Empirical studies of programmers: Second workshop* (G. M. Olson, S. Sheppard, and E. Soloway, eds.).

20. D. A. Scanlan (1989) Structured Flowcharts Outperform Pseudocode: An Experimental Comparison. *IEEE Software* **6**, 28-36.

21. I. Vessey & R. Weber (1986) Structured Tools and Conditional Logic: An Empirical Investigation. *Communications of the ACM* **29**, 48-57.

22. P. Mulholland (1997) Using a Fine-grained Comparative Evaluation Technique to Understand and Design Software Visualization Tools. In: *Empirical Studies of Programmers: Seventh Workshop* (S. Wiedenbeck and J. Scholtz, eds.) ACM Press, New York.

23. P. Mulholland (1998) A Principled Approach to the Evaluation of Software Visualization: A Case-study in Prolog. In: *Software Visualization: Programming as a multi-media experience* (J. Stasko, J. Domingue, M. Brown, and B. Price, eds.) MIT Press, Cambridge, MA.

24. D. Kurlander (1993) Chimera: Example-based Graphical Editing. In: *Watch What I Do: Programming by Demonstration* (A. Cypher, ed.) MIT Press, Cambridge, MA.

25. H. Lieberman (1992) Dominos and Storyboards: Beyond Icons on Strings. Presented at IEEE Conference on Visual Languages, Seattle.

26. F. Modugno, A. T. Corbett, & B. A. Myers (1996) Evaluating Program Representation in a Demonstrational Visual Shell. In: *Empirical Studies of Programmers: Sixth Workshop* (W. D. Gray and D. A. Boehm-Davis, eds.) Ablex, Norwood, NJ.

27. C. Traynor & M. G. Williams (1997) A Study of End-user Programming for Geographic Information Systems. In: *Empirical Studies of Programmers: Seventh Workshop* (S. Wiedenbeck and J. Scholtz, eds.) ACM Press, New York.

28. P. Whalley (1992) Making Control Technology Work in the Classroom. *British Journal of Educational Technology* **23**, 212-221.

29. J. Taylor (1990) Analysing Novices Analysing Prolog: What Stories do Novices Tell Themselves About Prolog? *Instructional Science* **19**, 283-309.

30. S. Papert (1993) *Mindstorms: Children, Computers and Powerful Ideas* Harvester Wheatsheaf, Hemel Hempstead, UK.

31. S. N. K. Watt (1998) Syntonicity and the Psychology of Programming. Presented at Psychology of Programming Interest Group, The Open University, UK.

32. H. A. Simon (1975) The functional equivalence of problem solving skills. *Cognitive Psychology* **7**, 268-288.

33. C. Lewis, C. Brand, G. Cherry, & C. Rader (1998) Adapting User Interface Design Methods to the Design of Educational Activities. Presented at Human Factors in Computing Systems, CHI '98, Los Angeles.

34. H. Kahney (1993) *Problem Solving: Current Issues. Second Edition* Open University Press, Buckingham, UK.

35. D. A. Norman (1993) *Things that Make us Smart: Defining Human Attributes in the Age of the Machine* Addison-Wesley, Reading, MA.

36. L. J. Bannon (1991) From Human Factors to Human Actors: The Role of Psychology and Human-Computer Interaction Studies in System Design. In: *Design at Work: Cooperative Design of Computer Systems* (J. Greenbaum and M. Kyng, eds.) Lawrence Erlbaum, Hillsdale, NJ.

37. K. Kutti (1996) Activity Theory as a Potential Framework for Human-Computer Interaction. In: *Context and Consciousness: Acitivity Theory and Human-Computer Interaction* (B. A. Nardi, ed.) MIT Press, Cambridge, MA, pp. 17-44.

```
ROVER
  |
  ---LIKES--->FIDO
  |
  ---CHASES--->CATS
  |
  ---DRINKS--->WHISKY
```

Figure 1. The triplet structure of the SOLO programming language.

Requirements	Issues	Design decisions
Cognitive modelling language for psychology students	Clear relation to the Information Processing framework and cognitive architectures	*Hank architecture comprising*
	SOLO — usable but tied to triplets, too limited Prolog — more powerful but masks control flow, inaccessible	*Fact cards*
Usable by non-programmers	Usefulness of grid-based layouts — noPumpG, CoCoa, Agentsheets, spreadsheets	
Usable in groups	Spreadsheets — good for group work	
Showing the process	Spreadsheets — good for declarative understanding and pattern matching (at the expense of computational structure) Flowcharts — good at showing control flow (but at the expense of data flow)	*Instruction cards*
	Software visualisation — good at showing process, but need to support mapping, reviewing, and testing Comic strips and Dominos in Programming by Demonstration	*Storyboards*
Usable on paper	Children produce the answer they want when working on paper 'real world' bugs Syntonicity — identifying with the program	*Executive*

Figure 2. Hank design overview.

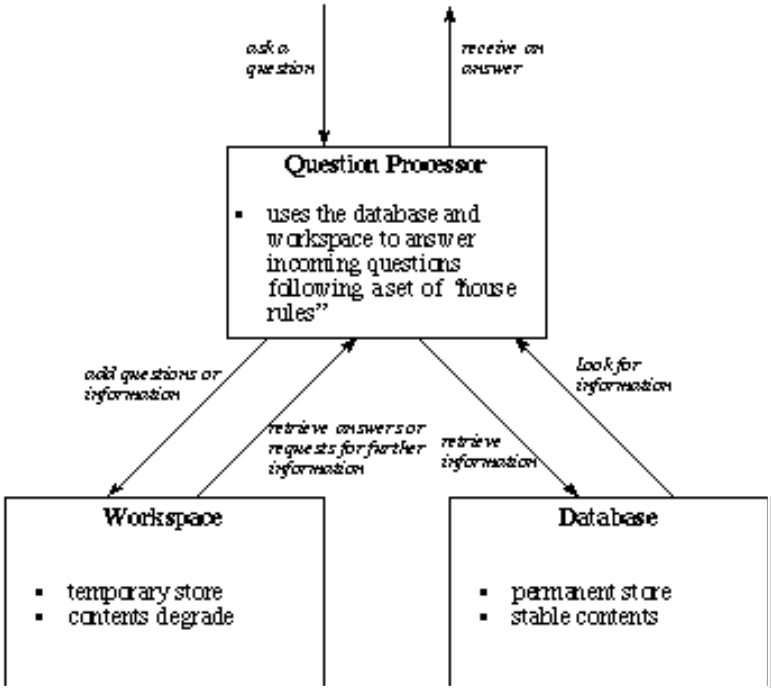
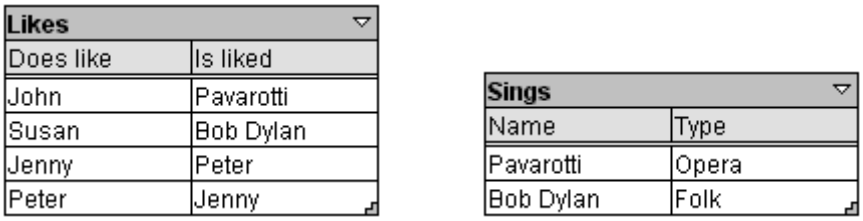


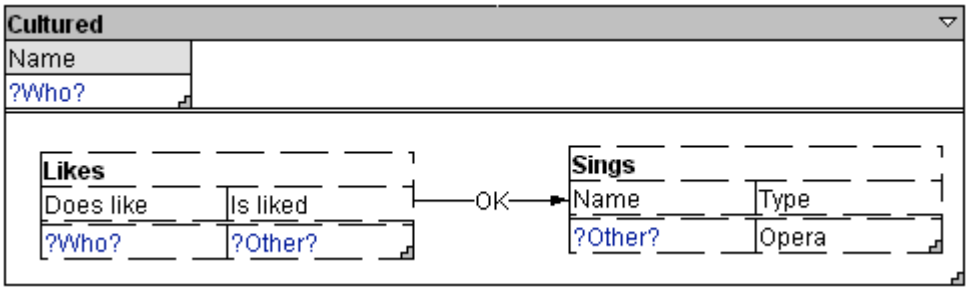
Figure 3. The Hank architecture.



Likes	
Does like	Is liked
John	Pavarotti
Susan	Bob Dylan
Jenny	Peter
Peter	Jenny

Sings	
Name	Type
Pavarotti	Opera
Bob Dylan	Folk

Figure 4. Hank fact cards.



Cultured

Name
?Who?

Likes	
Does like	Is liked
?Who?	?Other?

OK

Sings	
Name	Type
?Other?	Opera

Figure 5. A Hank instruction card.

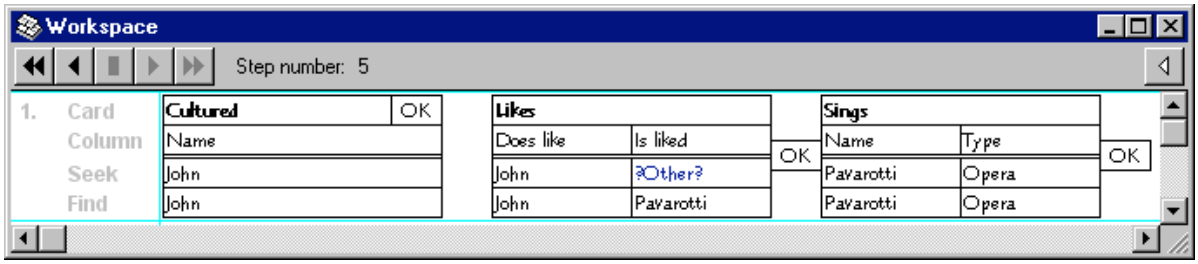


Figure 6. A Hank storyboard.

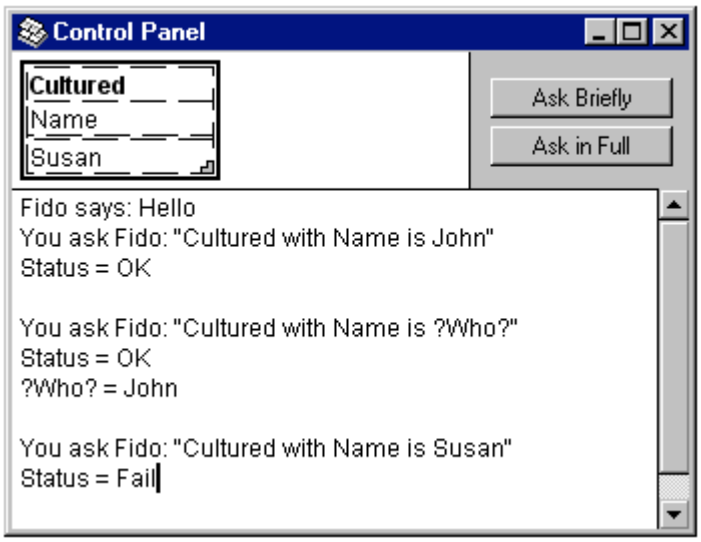


Figure 7. The Hank control panel.

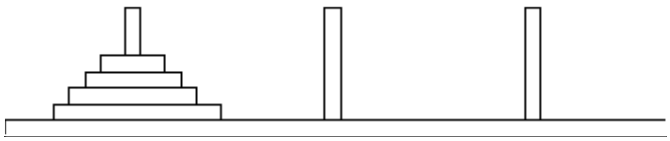


Figure 8. The Towers of Hanoi puzzle.

State		
Ring	Peg	Upon
Ring 4	Peg 1	Ring 3
Ring 3	Peg 1	Ring 2
Ring 2	Peg 1	Ring 1
Ring 1	Peg 1	Table

Figure 9. The Hank fact card representing the state shown in Figure 8.

P1. State = Problem-solved → Halt
P2. State = Done, Goal = Move(Pyramid(k), A)
→ Delete(STM), Goal ← Move(Pyramid(k – 1), A)
P3. State = Can, Goal = Move(Pyramid(k), A)
→ Delete(STM), Move(k, P(k), A)
P4. State = Cant, Goal = Move(Pyramid(k), A)
→ Delete(STM), Goal ← Move(Pyramid(k – 1), O(P(k),A))
P5. Goal = Move(Pyramid(k), A) → Test(Move(k, P(k), A))
P6. else → Goal ← Move(Pyramid(n), Goal-peg)

Figure 10. Production system for the goal recursion strategy, from [32].

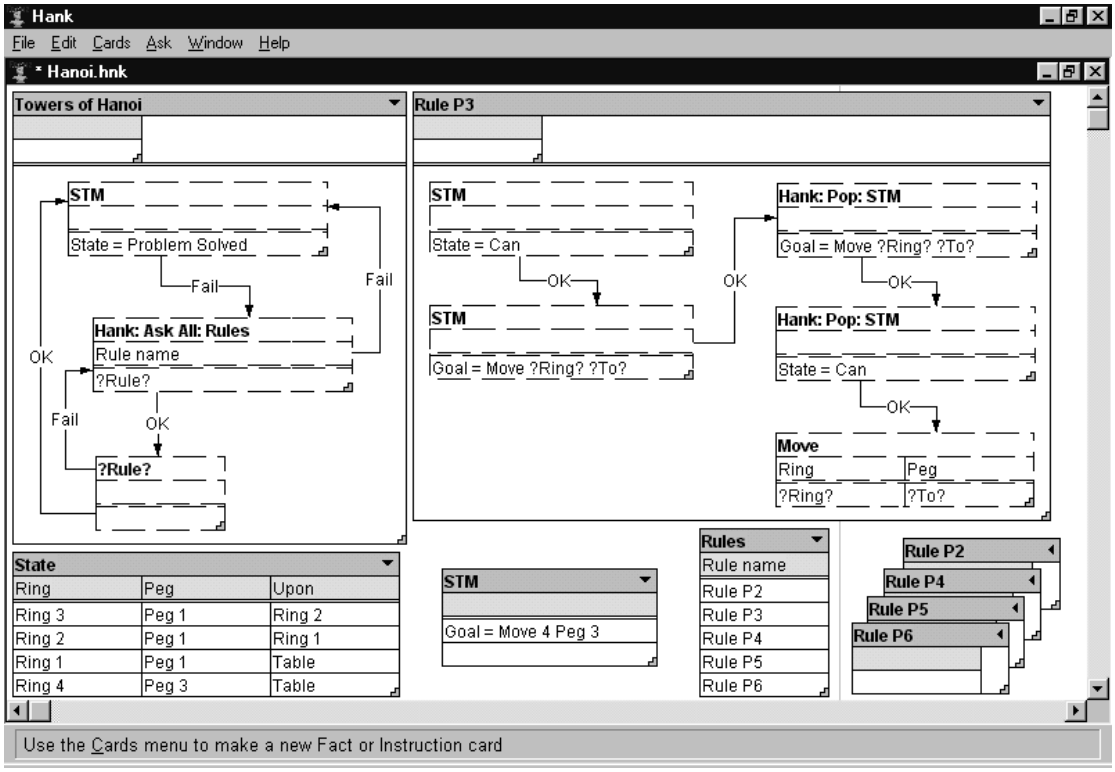


Figure 11. Screen display of part of the Towers of Hanoi model.

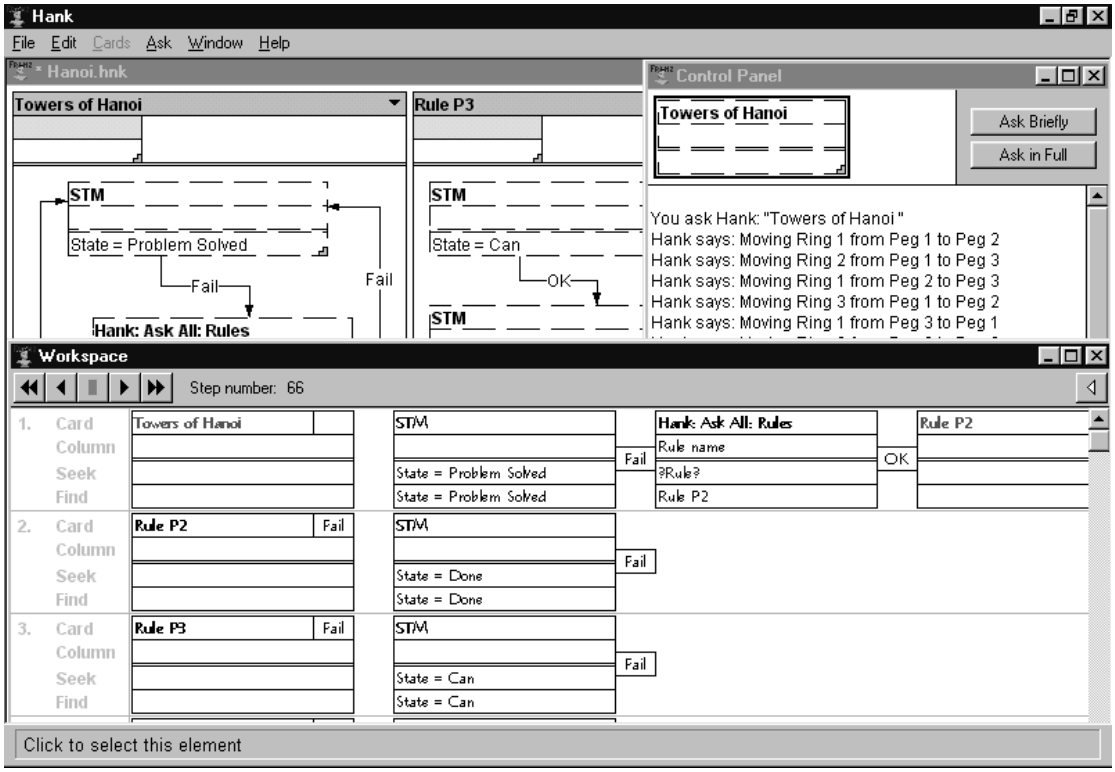


Figure 12. The Towers of Hanoi model, showing the workspace and the control panel.

Actor: John
Act: ATRANS
Object: necklace
Direction TO: Mary
Direction FROM: John

Schema	
Slot	Value
Actor	John
Act	ATRANS
Object	necklace
Direction TO	Mary
Direction FROM	John

Figure 13. On the left, a typical schema representation from [1] and (right) how the schema could be represented in Hank.

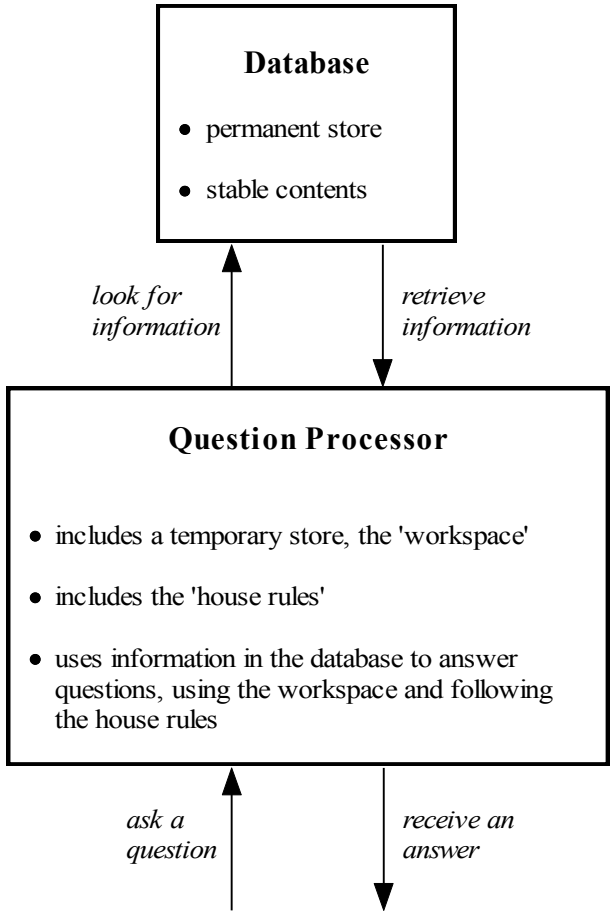


Figure 14. Revised Hank architecture.