Software Visualization Based KBS Validation

John Domingue

Knowledge Media Institute,
The Open University, Walton Hall,
Milton Keynes, MK7 6AA.
J.B.Domingue@open.ac.uk

Abstract. The validation of a Knowledge Based System (KBS) involves comparisons between an external reference model and a system's component parts. In this paper I describe how such comparisons can be aided by the application of software visualization technology. Software visualization is the use of filmcraft, cartoon animation and graphic design techniques to display data structures, programs, and algorithms. The described approach eases the task of mapping between the comparates by the use of dynamic code, problem solving architecture, and domain oriented visualizations of KBS execution.

1. Introduction

Software Visualization (SV) [Price et al., 1993] is the use of filmcraft, cartoon animation and graphic design techniques to display data structures, programs, and algorithms. Software visualization is basically the unification of algorithm animation and program visualization, two concurrent developments from the 1980s. Algorithm animations (e.g. [Brown, 1988], [Stasko, 1990]) are high level characterisations of how data is manipulated during a program execution. Program visualization systems (e.g. [Domingue & Eisenstadt, 1989]) display graphical representations that are more tightly coupled with a program's code or data and show more or less faithful representations of the code as it is executing.

The validation of a Knowledge Based System (KBS) involves the comparison of a model residing within the KBS against an external reference model. A variety of techniques exist for carrying out the comparisons. O'Keefe et al., [1987] described a set of qualitative validation techniques which could be aided by the use of graphical interfaces. These included:

- Face Validation - project team members, potential users and domain experts compare system performance against human performance.
- Turing Tests - inspired by [Turing, 1950] the tests are as above but carried out blind, the evaluators do not know if they are evaluating a human expert or a KBS. Turing tests prevent evaluators biasing against a KBS, as was found to be the case in the evaluations of MYCIN [Gaschnig et al., 1983].
- Subsystem Validation - subsystems are validated individually. The advantages of this approach are:

    i) validation is carried out along with development,

    ii) subsystems are less complex and therefore easier to validate,

    iii) error detection is much easier since errors are localised,

  the disadvantages are:

    i) it may not be possible to observe the input-output behaviour of a subsystem,

    ii) successfully validating each subsystem does not imply overall system validity.
- Sensitivity Analysis - systematically changing system input, variable values and parameters over some range of interest and observing the effect on system performance.

In this paper I describe how software visualization technology has been used to support the above validation techniques. In particular I shall focus on an extension to the Turing test inspired validation technique.

The rest of this paper is structured as follows. In the next section I describe the SV based validation approach using the Sisyphus I [Linster, 1992] example. The final section contains some discussions and conclusions.

2. The Validation of Sisyphus I

2.1. Sisyphus I The Room Allocation Problem

The Sisyphus I problem [Linster, 1992] is one of the standard problems used within the knowledge acquisition community to compare KBS systems and methodologies. The problem is to assign a set of researchers from a laboratory to a set of rooms within a new building. Various constraints are placed on the allocation of rooms. The constraints are:

i) the head of the group should be in a central office,

ii) the secretaries should be close to the head of the group,

iii) the project managers should have maximum access to the secretaries and the head of the group,

iv) smokers and non-smokers should not share an office,

v) as far as possible researchers sharing an office should be working on different projects.

The implementation of this example is described in detail in [Motta et al., 1994]. I shall now describe our approach to validation using this example.

2.2. The Expert Scripted Visualization

Enabling domain experts and KBSs to compete ‘on a level playing field’ in a Turing test is a non-trivial problem. Gaschnig et al. [1983] described several problems:

> “Since the typescript protocol generated by an expert system cannot very well be disguised to resemble human dialogue, the desire to blind a study limits the amount of evidence that can be made available to the panel of expert judges...” p. 251.

> “The experts engaged in evaluating system performance must have full access to all aspects of behaviour, so that they can push and probe, looking for weaknesses and deficiencies. This would seem to rule out blinded, comparative studies as an appropriate framework for expert-system evaluation...” p. 252.

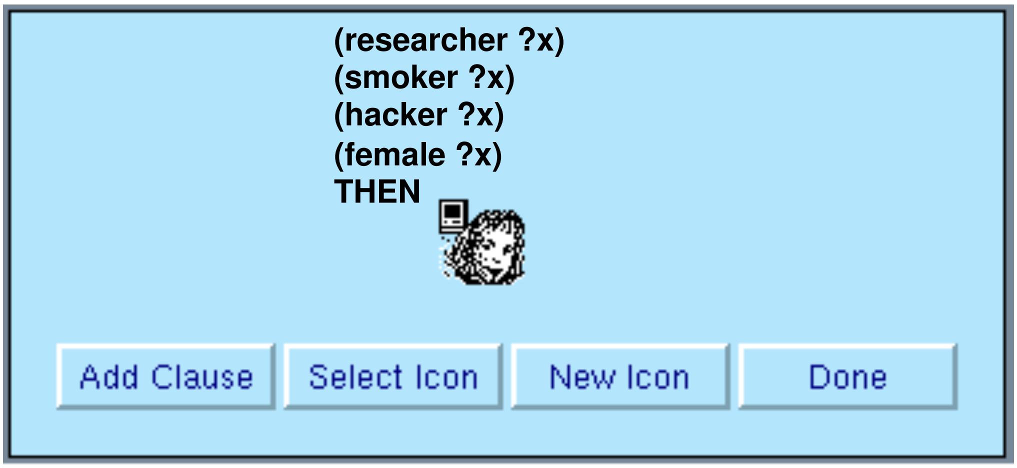

Figure 1. A snapshot of the menu used to create mappings between concepts and graphical icons.

The solution to the problem has been to use SV as a common rich description language for the expert and KBS. In conjunction with a Knowledge Engineer (KE) the domain expert can create an Expert Scripted Visualization (ESV). I shall now describe how this is done. Using a special purpose menu the KE creates a mapping between domain concepts and/or instances to graphical icons. This can be carried out during knowledge elicitation (using an enhanced protocol editor) or at a later stage. The menu in figure 1 above, taken from the Sisyphus I example, indicates that female researchers who smoke and program are to be represented by the  icon.

The full set of mappings is shown below:

-  or  - researcher,
-  or  - smoker,
-  or  - hacker,
-  or  - hacker and smoker,
-  - manager,
-  - secretary,
-  - head of group.

- room.

Once a set of mappings have been created a specialised Sisyphus I script editor can be created. A snapshot of one in use is shown in figure 2 below.

Using the above editor a scenario can be created by dragging the appropriate icons from the palette on the left of the window. The domain expert can then carry out a task, in this case a room allocation, by direct manipulation. The manipulations can be recorded and replayed later on. An SV supported Turing test thus involves the comparison of an ESV against a KBS domain visualization.

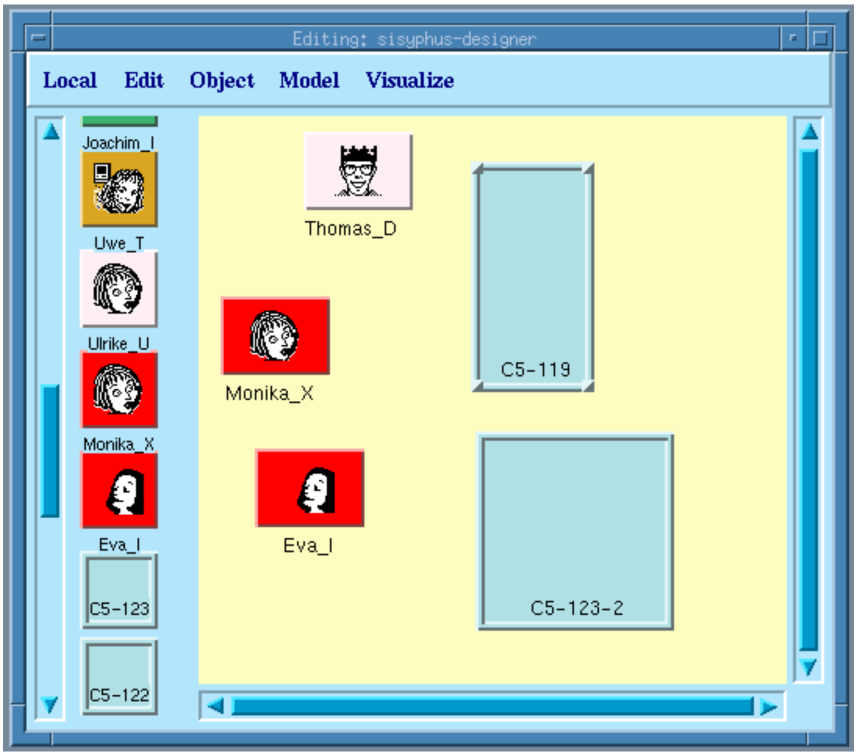

Figure 2. A snapshot of a Sisyphus I script editor. The palette on the left is a well containing all the YQT researchers and the rooms.

2.2.1. The Sisyphus I Domain Visualization

By comparing the ESV (the final state of the top row is shown in figure 3) with the KBS generated domain visualization (figure 4) the KE notices that the top rows are different - the fourth constraint has been violated by the KBS. The rooms C5-120 and C5-121 contain a smoker and non-smoker. Notice that a mismatch between expert behaviour and the KBS behaviour is extremely easy to see using the domain visualization.

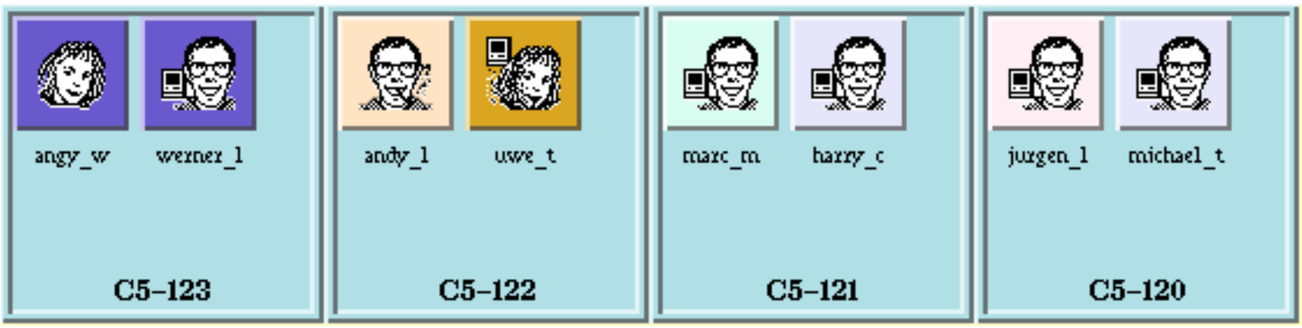

Figure 3. The top row of the expert scripted visualization.

Figure 4. The top row the KBS driven domain visualization

Having determined that there is a mismatch between expert and KBS behaviour the KE now wants to pinpoint the part of the design responsible. To aid this knowledge level visualizations of the design are provided using a Problem Solving Architecture (PSA) document. A PSA specifies the control and data flow between tasks and is created when a KBS is designed. A PSA visualization is obtained by highlighting the currently executing task in red (dark grey in this document). The task 'guilty' of placing the smokers and non-smokers together can be tracked down in the following way. The KE adds a pause point, for the 'assign to room' event, to the angy_w icon in figure 4 above. This is achieved by selecting the angy_w icon and choosing 'Pause' from the 'Operations' menu. The KE then replays the PSA visualization.

Figure 5 below shows the PSA diagram when the replay halts. I shall now briefly explain the semantics of the diagram. Tasks are represented by ovals and rôles by rectangles (note that I distinguish between rôles in a PSA and Sisyphus roles with a circumflex). Tasks specify knowledge level activities. A rôle represents some segment of task input or output data (defined in a concept/instance or relation module). The cardinality of a rôle is indicated by a numeral in the top right corner of the rectangle. The solid lines in the diagram are control flow links and the dashed lines are data flow links. Duplications or aliases (two nodes representing the same task or role) are indicated by the use of an italic font. PSA diagrams can also contain control constructs. Three are present in the figure below. The hexagonal node is a choice control construct. In a choice construct control flow passes to one of the subtasks depending on the value of the input role. The rounded rectangle labelled 'Return1' is a return construct. The execution of a return construct passes control back to the root task (shown as a double oval). The remaining rounded rectangle is a sequence control construct. A sequence construct is used for aesthetic purposes, to make diagrams grow vertically rather than horizontally.

I shall now explain the design specified by the diagram. In order to carry out the design allocation task we choose a role using the role-order and the current-role. The new current-role determines which assignment task (allocate-head-of-group, allocate-secretaries etc.) is chosen. We keep doing this until the current-role becomes empty. The rôles current-role and role-order are defined in the relations module.

The domain visualization (partly shown in figure 4) and the PSA visualization are tied together. Replaying one replays the other. As the allocate-researchers task is highlighted (as depicted in figure 5) the eight researchers shown in figure 4 slide into the rooms. This means that the bug is within the implementation code attached to the allocated-researchers task. It is possible to examine the execution of the allocate-researchers task at the implementation level. This is achieved by selecting the task and choosing 'Show Code Level Visualization' from the 'Operations' menu. Unfortunately we do not have space to describe the code level visualizations here. Suffice to say that a set of visualizations are provided for the main knowledge representation languages such as OPS5 (see [Domingue & Eisenstadt, 1989]), Prolog (see [Eisenstadt & Brayshaw, 1988]) and Lisp.

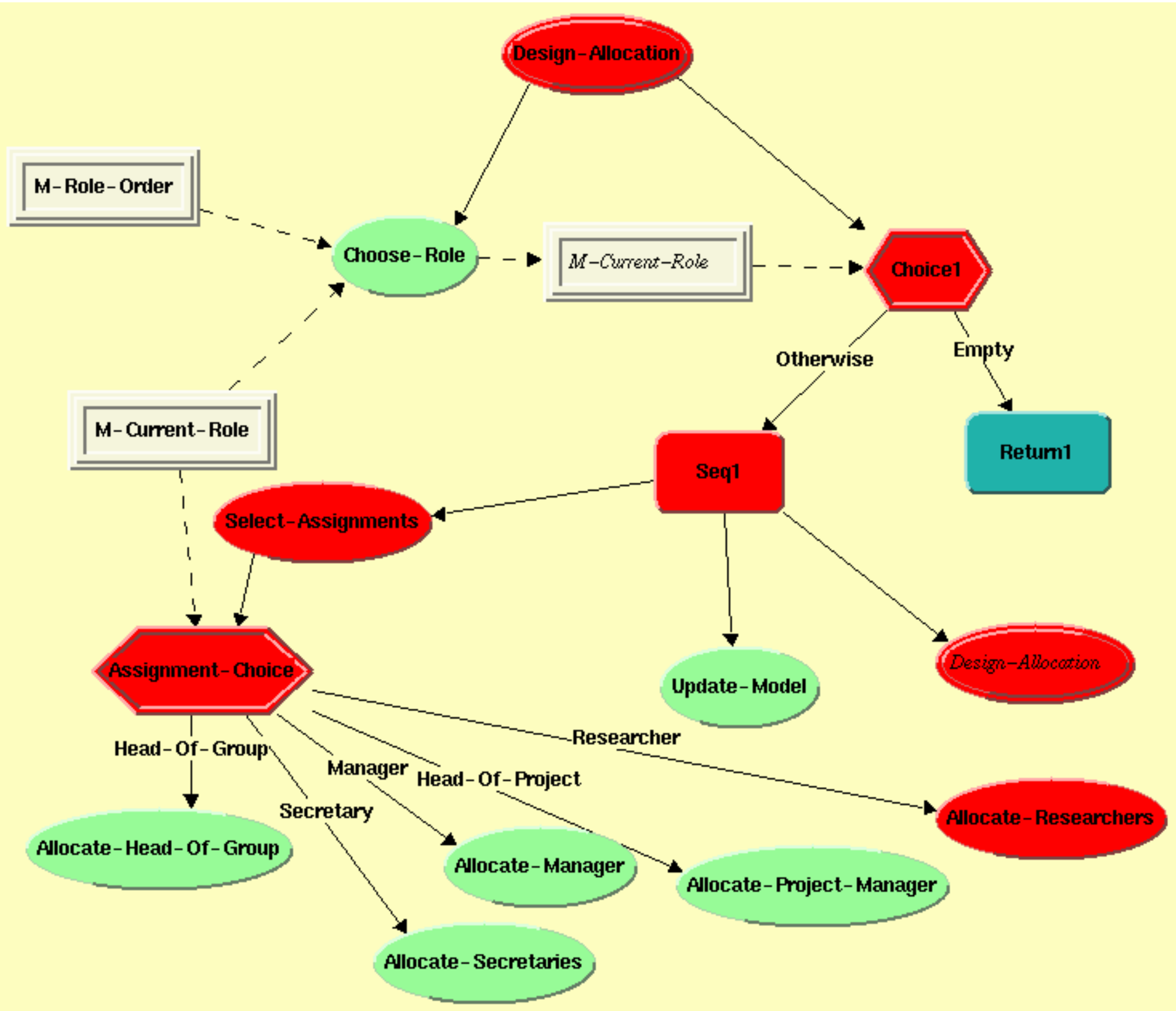

Figure 5. The PSA diagram after replaying to the paused point.

3. Discussion and conclusions

3.1. Visualization Support Provided

The problem for a KE, in carrying out validation, is to map between a source expert and the various models within a KBS. The approach here has been to reduce the mapping effort required by extensive use of SV techniques providing synchronised problem solving architecture, domain and code views onto a KBS. In the example we saw how the comparison between an expert scripted and KBS driven domain visualization enabled us to see quickly a mismatch between a specified behaviour (the list of constraints) and a system's behaviour. Locating the part of the code responsible for this is aided by:

- the fact that expert and KBS behaviour have a common rich representation,
- the fact that the visualizations are synchronised,
- the ability to add pause points to the domain view,
- the ability to look at a 'task slice' of the code level execution, and
- the ability to view the code executions at coarse and fine-grained views.

A summary of the support the workbench provides in design and validation is depicted in figure 6 below.

I shall now discuss how SV can aid in each of the validation types outlined within the introduction.

3.2. Face Validation.

We can see from the example that with SV technology face validation becomes relatively easy. Simple invalid behaviours such as a researcher moving to one room then to another, or a particular role not being allocated become obvious. Some quite tricky concepts become explainable. For example, from the expert protocol it is not obvious what the concept of 'maximum access to' means in the third constraint. I would argue that the definition used in the design model, which uses four rules, would be harder for a domain expert to understand than just observing the domain visualization.

3.3. Subsystem Validation.

SV can aid in subsystem validation. Using the design document we can fast forward to a particular task. If a task is executed more than once we can choose which execution to view in detail. Navigation facilities are also provided at the implementation level. This eliminates the first problem outlined by O'Keefe (see introduction), that of observing the input-output behaviour of a subsystem.

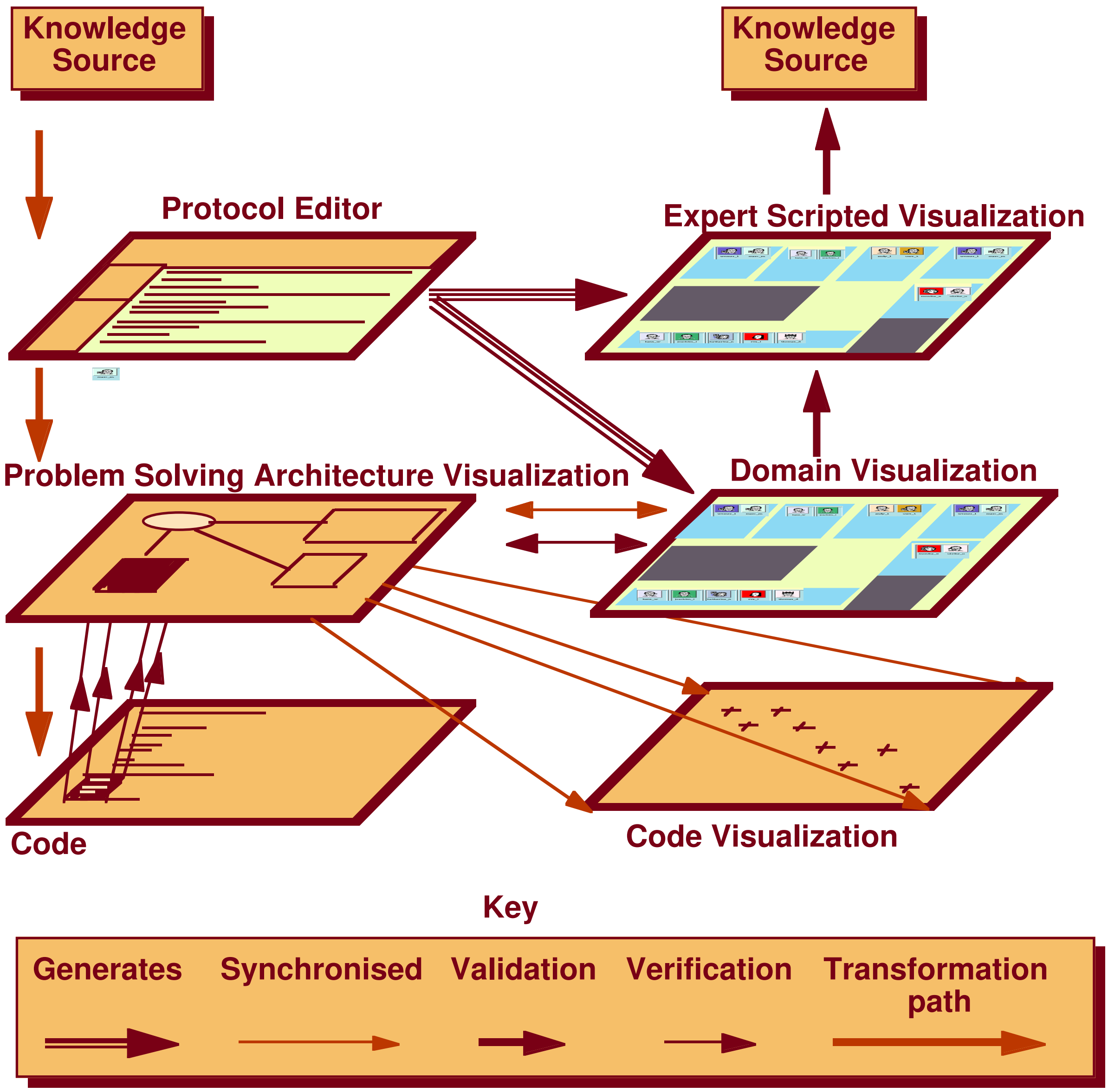

Figure 6. The SV based validation support provided by VITAL.

3.4. Sensitivity Analysis.

Sensitivity analysis involves the comparison of separate KBS executions. In previous work [Domingue, 1988; 1990] I showed how code level KBS executions could be compared. Two or more design visualizations can be compared by replaying the executions on the same design document and assigning a different highlighting colour to each one. The attached domain visualizations can then be viewed simultaneously (given adequate screen space).

3.5. Concluding Remarks

As Guida and Mauri [1993] say:

> "...it is straightforward to recognize that the ultimate goal of evaluation is KBS behaviour" p. 209

but

> "Measuring performance directly from the definition turns out to be a difficult task: in fact, it is hard to envisage the actual behaviour of a KBS and to compare it with the specifications." p. 210

The approach outlined in this paper enables KBS and expert behaviour to be compared easily because SV provides a common rich representation. It should also be noted that the construction of domain visualizations (whether expert or KBS driven) is relatively easy for KBS because domain knowledge is represented explicitly and can therefore be straightforwardly mapped into an iconic representation.

In addition to the Sisyphus I example we have used this approach on a telephone line assignment application (for a client of Andersen Consulting) and we have created visualizations for the Sisyphus II example [Yost, 1992].

With this approach I believe that KBS development is enhanced for the various involved communities in three ways. Firstly, systems builders benefit because problem areas can be identified more quickly. Secondly, communication between system builders, domain experts and end users are improved, thus increasing acceptability chances. Finally, researchers are able gain new insights into how a KBS reasons.

References

Brown, M.H. (1988). *Algorithm Animation.* ACM Distinguished Dissertations, MIT Press, New York.

Domingue, J. (1988). TRI: The Transparent Rule Interpreter. Proceedings of Expert Systems'88 in Shadbolt, N. (Ed.) *Research and Development in Expert Systems VII,* Cambridge University Press, December, 1988.

Domingue, J. and Eisenstadt, M. (1989) A New Metaphor for the Graphical Explanation of Forward Chaining Rule Execution. In Proceedings of The Twelfth International Joint Conference on Artificial Intelligence (Detroit, MI), pp. 129-134.

Domingue, J. (1990). Compressing and Comparing Metric Execution Spaces. In D. Diaper, D., Gilmore, G. Cockton, G. and B. Shackel (Eds.) Proceedings of the 3rd IFIP Conference on Human-Computer Interaction, Cambridge, *Human-Computer Interaction: Interact'90,* North-Holland, August, 1990.

Eisenstadt, M. and Brayshaw, M. (1988). The Transparent Prolog Machine (TPM): an execution model and graphical debugger for logic programming. *Journal of Logic Programming*, 5(4), pp. 277-342.

Gaschnig, J. Klahr, P. Pople, H. Shortliffe, E. and Terry, A. (1983). Evaluation Of Expert Systems: Issues And Case Studies. In Hayes-Roth, F., Waterman, D. A., and Lenat D. B. (Eds.) *Building Expert Systems* Addison-Wesley, New York. pp. 241-280.

Guida, G., and Mauri, G. (1993). Evaluating Performance and Quality of Knowledge-Based Systems: Foundation and Methodology. *IEEE Transactions on Knowledge and Data Engineering.* Vol. 5. No. 2, April, 1993.

Linster M. (Ed.) (1992). Sisyphus '92: Models of problem solving. GMD report 630, GMD Sankt Augustin.

Motta, E., O'Hara, K., and Shadbolt, N. (1994). Grounding GDMs: A Structured Case Study. *International Journal of Human-Computer Studies,* 40, March, pp. 315-348.

O'Keefe, R., Balci, O. and Smith, E. P. (1987). Validating Expert System Performance. *IEEE Expert,* 4(11) Winter, 1987.

Price, B. A., Baecker, R. M. and Small, I.S. (1993). A Principled Taxonomy of Software Visualization. *Journal of Visual Languages and Computing* 4 (3), pp. 211-266.

Stasko, J. T. (1990). Tango: A Framework and System for Algorithm Animation. *IEEE Computer*, 23(9) pp. 27-39.

Turing, A. M. (1950). Computing machinery and Intelligence. *Mind* pp. 433-460.

Yost, G. R. (1992). Configuring Elevator Systems. Workplace Integration Technologies Group, Digital Equipment Corporation, 11 Locke Drive (LM02/K11), Marlboro, MA 01752. December. Unpublished.