Building Knowledge Models Using KSM

Jose Cuena, Martin Molina

Department of Artificial Intelligence, Technical University of Madrid,
Campus de Montegancedo S/N, Boadilla del Monte 28660, Madrid, SPAIN
{jcuena,mmolina}@dia.fi.upm.es
http://www.isys.dia.fi.upm.es/ksm

Abstract. This paper describes the operation with the software environment called KSM (Knowledge Structure Manager) that supports a methodology for building and reusing knowledge models. The methodology is a useful tool for developers who need to build large and complex knowledge models in real world projects. The enviroment helps developers in applying the whole methodology to build the final system and it also assists end users during the operation and maintenance of existing knowledge models. The paper introduces first briefly the knowledge modeling concept of KSM and, then, the operation with the environment is described and illustrated with some examples of screens. Finally, the paper presents some technical details about the software architecture of KSM .

1. INTRODUCTION

KSM (Knowledge Structure Manager) [Cuena, Molina, 94] [Cuena, Molina, 96] is a software environment that supports a methodology for building and reusing knowledge models. The methodology is a useful tool for developers who need to build large and complex knowledge models in real world projects. The enviroment helps developers in applying the whole methodology in order to build the operational version of the final system and it also assists end-users during the operation and maintenance of existing knowledge models. KSM was one of the results of our research group ISYS (Intelligent Systems) in the Artificial Intelligence Department of the Technical University of Madrid. It includes experience of the last five years in some knowledge engineering projects about different domains such as bank management decision support, flood emergencies in rivers (CYRAH [Cuena et al., 92] and SIRAH systems [Alonso et al., 90]), chemical plant monitoring (FIABEX system) or urban traffic control (TRYS system [Cuena et al. 96]). KSM has been influenced by parallel knowledge engineering research such as the concept of generic task proposed by Chandrasekaran [Chandrasekaran, 86] and the knowledge level concept proposed by Newell [Newell, 82]. It includes and extends some common ideas of other similar approaches of knowledge engineering methodologies and tools such as KADS [Wielinga, Breuker, 86], PROTEGE-II [Puerta et al., 93] and KREST [McIntyre, 93], [Steels, 90]. Presently, KSM and its methodology is an effective tool for knowledge modelling that is being used to develop other different projects.

The paper describes first a brief introduction of the knowledge modeling concept of KSM (section 2). Then, section 3 describes the operation with the environment using some examples of screens. Finally, section 4 includes some technical characteristics about the software architecture of the environment.

2. GENERAL VIEW OF THE KSM KNOWLEDGE MODEL

Basically, to formulate a knowledge model in KSM, three perspectives are defined: (1) the knowledge-area perspective, which plays the role of the central structure of the model as a structured collection of knowledge bodies, (2) the task perspective, similar to the task layer of KADS and (3) the vocabulary perspective, which includes the basic terms shared by several knowledge modules.

The knowledge-area perspective is used for presenting a general image of the model where each module represents what is called *knowledge area*. In general, a knowledge area identifies a body of expertise that explains a certain problem-solving behaviour of an intelligent agent. Typically, a knowledge area is associated to a professional skill, a qualification or speciality of an expert. The whole knowledge model is a hierarchical structure of knowledge areas in such a way that there is a top-level area representing the entire model. This area is divided (using the part-of relation) into other more detailed subareas that, in their turn, are divided into other simpler areas and so on, developing the whole hierarchy (where some areas may belong to more than one higher level area). A bottom level area is called *primary knowledge area* and corresponds to an elementary module that may be directly operationalized by using basic software tools.

Knowledge areas are not passive modules but, in general, they can provide different services represented by a set of tasks. The task perspective presents a functional description for each task using a tree of task-method-subtasks. A *task* defines a goal to achieve (with a set of inputs and a set of outputs) and the *method* describes how to carry out the task. In KSM methods are formulated using a particular language called *Link* (supported by an interpreter at run time). This language allows to represent the line of reasoning of problem-solving methods with two main parts: the data flow section, that defines how tasks are connected, and the control flow section, that uses rules to establish the execution order of tasks. KSM uses an additional descriptive component called *knowledge unit* that is defined by the triple <*A*, *K*, *T*>, where *A* is the area associated to the knowledge unit, *K* is the set of subareas in which *A* is decomposed into and *T* is the set of tasks provided by *A*. The purpose of the knowledge unit is to serve as a package that encapsulates the components associated to a knowledge area.

Finally, the vocabulary perspective is formulated with a set of components called conceptual vocabularies. A *conceptual vocabulary* defines a basic terminology used by several knowledge areas. A vocabulary is not a description of the whole domain knowledge, but it defines a partial view including the basic terms that are common to different knowledge bases. In KSM vocabularies are formulated using

a particular language called *Concel* that uses a concept-attribute-facet representation together with an organization in classes-subclasses-instances.

In order to operationalize a knowledge model defined by the previous three perspectives, KSM provides computable constructs implementing basic problem-solving techniques that are called *primitives of representation*. The knowledge level model is operationalized by associating these components to knowledge areas. A primitive may be viewed as a generalization of the concept of shell, considering also non knowledge based techniques. Each primitive include a knowledge acquisition user interface, a set of inference procedures and explanation facilities. In order to produce the operational version of the knowledge model, the developer associates a copy of a primitive to each primary knowledge area. KSM has an open library of such primitives to support the knowledge model. The library can be extended with other more specific primitives. Thus, the use of an open library allow to manage a multi-representation environment where it is possible to select the most appropriate technique for each case. This is particularly important in real systems where efficiency must be taken into account.

The structure of knowledge-areas, tasks and vocabularies (with the associated primitives of representation for operationalizing) is called *generic* model, given that it is general and reusable. To develop a model for a particular domain the developer creates a quasi-isomorphic structure of knowledge areas specialized in the domain as an instantiation of the general description. For each generic knowledge area there will be one or more domain knowledge areas, following the same relations established by the generic model. The developer particularizes the domain structure writing particular knowledge bases (using the set of knowledge acquisition facilities provided by primitives), creates domain conceptual vocabularies and he/she also may redefine at domain level the generic control knowledge defined in methods using the Link language. After this process, the model is ready to be executed for solving problems. Then, KSM also assists the operator during the maintenance and the execution of the final model.

2. THE KSM OPERATION

This section describes the KSM operation using examples of screens to illustrate how the user of KSM can create and edit knowledge models. Figure 1 shows the main window of the KSM environment that is presented when the application is started up. From left to right, the first button allows the user to quit the application. The second button, *primitives of representation*, is used to examine the list of basic building blocks provided by the library of KSM to operationalize a model. The following two buttons of the main window, called respectively *generic model* and *domain model*, are used to create or edit knowledge components at the corresponding levels (generic and domain). The last button, called *help*, provides a general explanation about the operation with the environment.

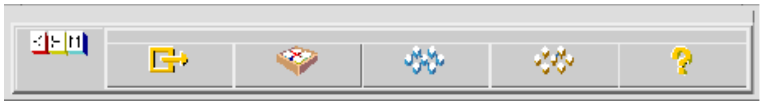

Figure 1: Main window of the KSM environment

The development of a knowledge model starts at generic level. The developer uses the options corresponding to the button *generic model*, to create (or edit) generic knowledge areas, tasks and vocabularies. One of these options, called *knowledge area view*, presents a window with a graphical view of the current structure of knowledge areas. Figure 2 shows an example of this window (on the left hand side). This window shows a partial view of a knowledge model for real-time decision support (this model considers a system that evolves over time and might present certain functional problems; the system's state is monitored periodically by using detectors that record basic magnitudes at significant locations, and it may be changed by acting on control devices or effectors which modify the behaviour of some components; examples of such a system are: a basin that can present floods at certain locations of rivers, a traffic network in a urban area where problems are traffic jams and the control actions may improve the traffic circulation, a chemical plant that can present certain malfunctions, etc.).

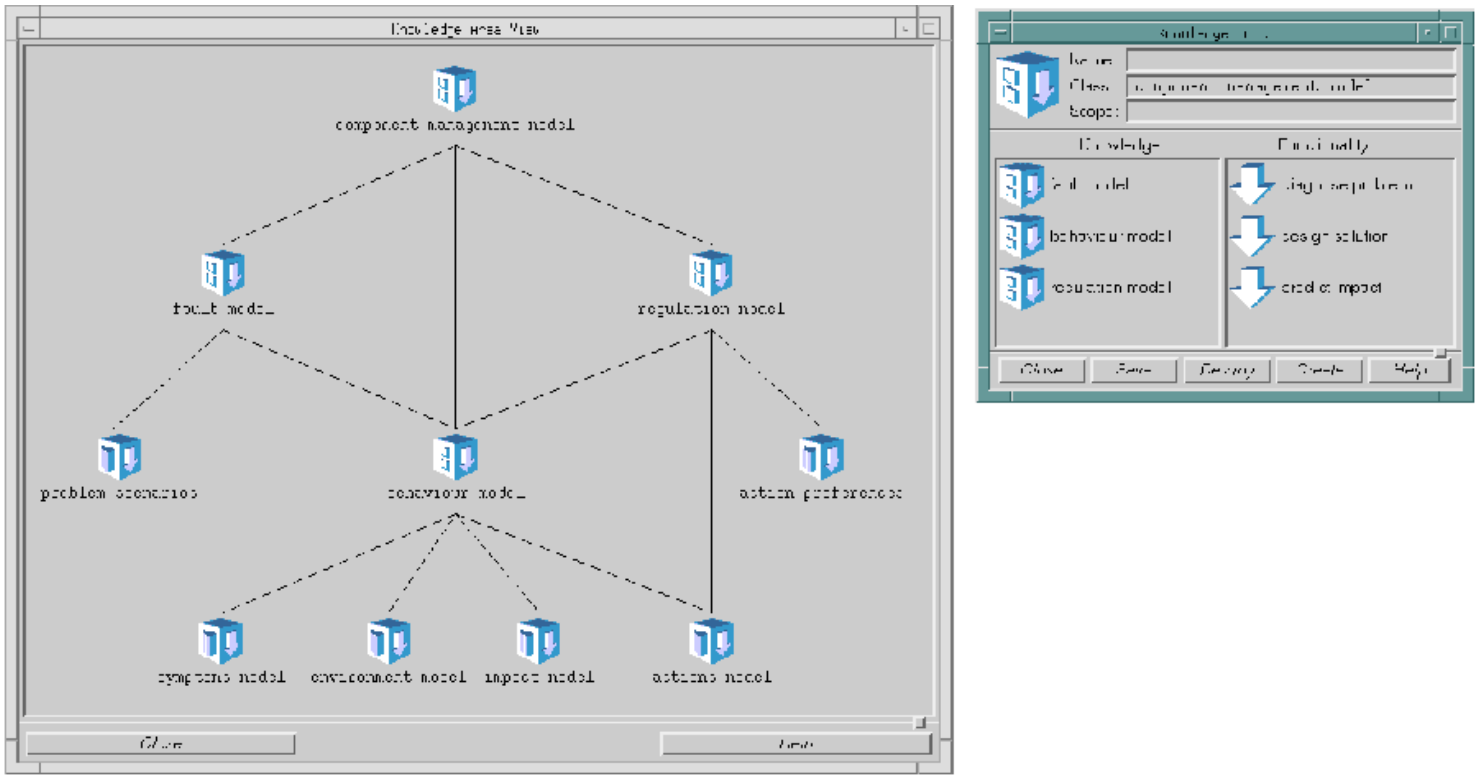

Figure 2: Example of knowledge area perspective of a generic knowledge model

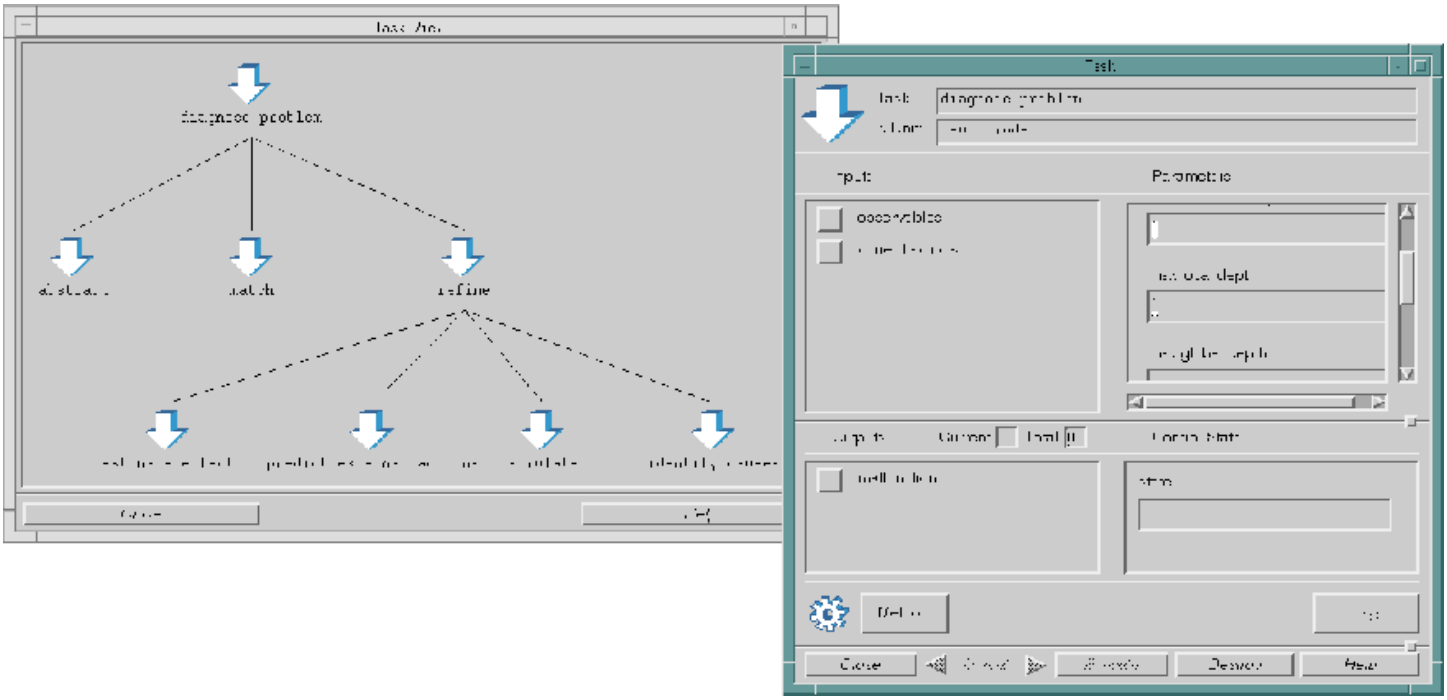

Figure 3: Example of task perspective of a generic knowledge model

The model of figure 2 includes a top-level knowledge area called component management model that represents the required knowledge to detect problems and propose solutions for a particular component of the system (e.g., a particular road of a traffic network for the case of the traffic domain). This area is divided into three areas: fault model, behaviour model and regulation model. In its turn, each of these three areas is divided into other simpler areas developing the whole hierarchy. At the bottom level there are primary areas that are not decomposed into other areas (problem scenarios, symptoms, model, etc.).

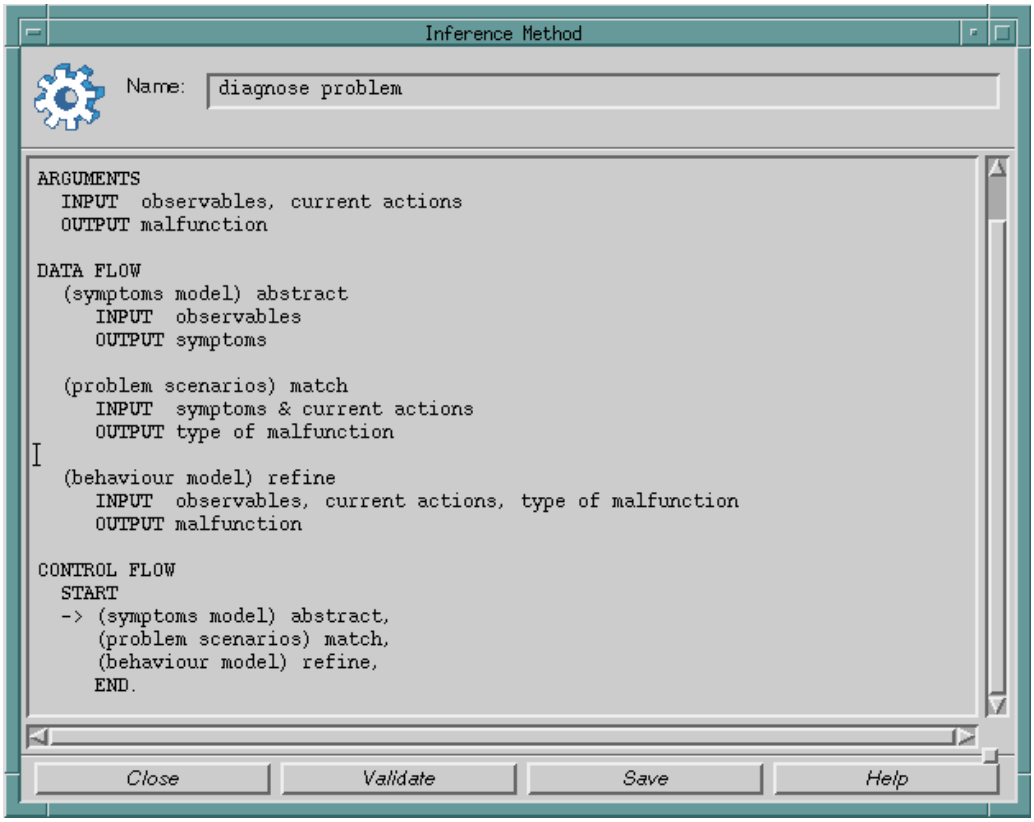

Figure 4: Example of method formulation using the Link language

The user can click an area to edit its contents. Doing so, a new window is presented (figure 2 shows on the right hand side an example of this window for the area called component management model). It presents more detailed information in the form of a knowledge unit. The left hand side of this window presents the sub-areas in which the area is divided (fault model, behaviour model and regulation model), and the right hand side includes the set of tasks provided by this unit (diagnose problem, predict impact and design solution). The window allows the user to delete or to modify this unit by creating new elements (sub-areas or tasks) using the buttons at the bottom.

The operator can also edit a particular task perspective of the model. In order to do that, first the user edits the corresponding knowledge unit that contains the task and, then, by clicking on the task, KSM presents a pop-up menu that allows to access a task tree. Figure 3 presents on the left hand side a window corresponding to the tree for the task diagnose problem (that belongs to the knowledge unit called fault model). This tasks is divided into three subtasks: abstract, match and refine. In its turn, the task refine is divided into four subtasks. By clicking on a particular task, KSM presents a new window with more detailed information. Figure 3 presents on the right hand side the window corresponding to the task diagnose problem. This window shows (on the lef hand side) the inputs of the task (observables and current actions) and the outputs (malfunctions). It shows also (on the right) control parameters to select modes of execution, and the control state which indicates the degree of success after the execution of the task. This window can be used (as it will be described later) to interactively execute particular tasks.

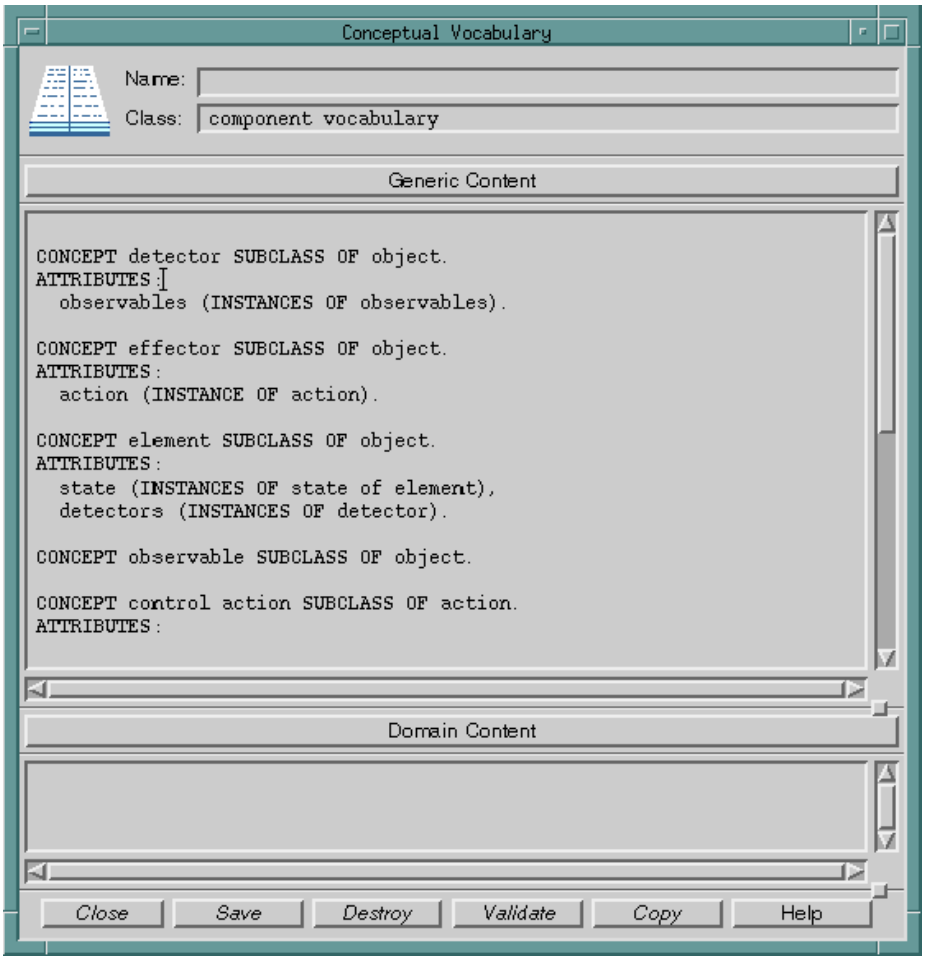

Figure 5: Example of conceptual vocabuary formulation using the Concel language

At the bottom left corner of the task window, the button called method allows the user to edit the method associated to the task. When clicking this button, KSM presents a new window corresponding to the method. Figure 4 shows an example of this window for the task diagnose problem. This window is like a text editor where the developer writes the method using the Link language. The buttons at the bottom are used to validate and compile the text and adecuate error messages are presented when the text is not correct. In the example, a particular version of the method heuristic-classification has been written to carry out the diagnose problem task. Global inputs are observables and current actions and the output is a malfunction. The method uses three subtasks: the task abstract (of the symptoms model), the task match (of the problem scenarios knowledge area) and the task refine (of the behaviour model). These tasks are connected as it is shown in the data flow section. For instance, the output of the task abstract (called symptoms) is input of the task match. In the control flow section of the Link language, the execution order is established using rules that include in the left hand side conditions about execution states. In this case, given that the three subtasks are executed in sequence, only a simple rule was required. This rule, at the left hand side includes as condition that it can be triggered at the beginning and, at the right hand side, it includes the sequence of the subtask execution: first abstract, second match and, finally, refine.

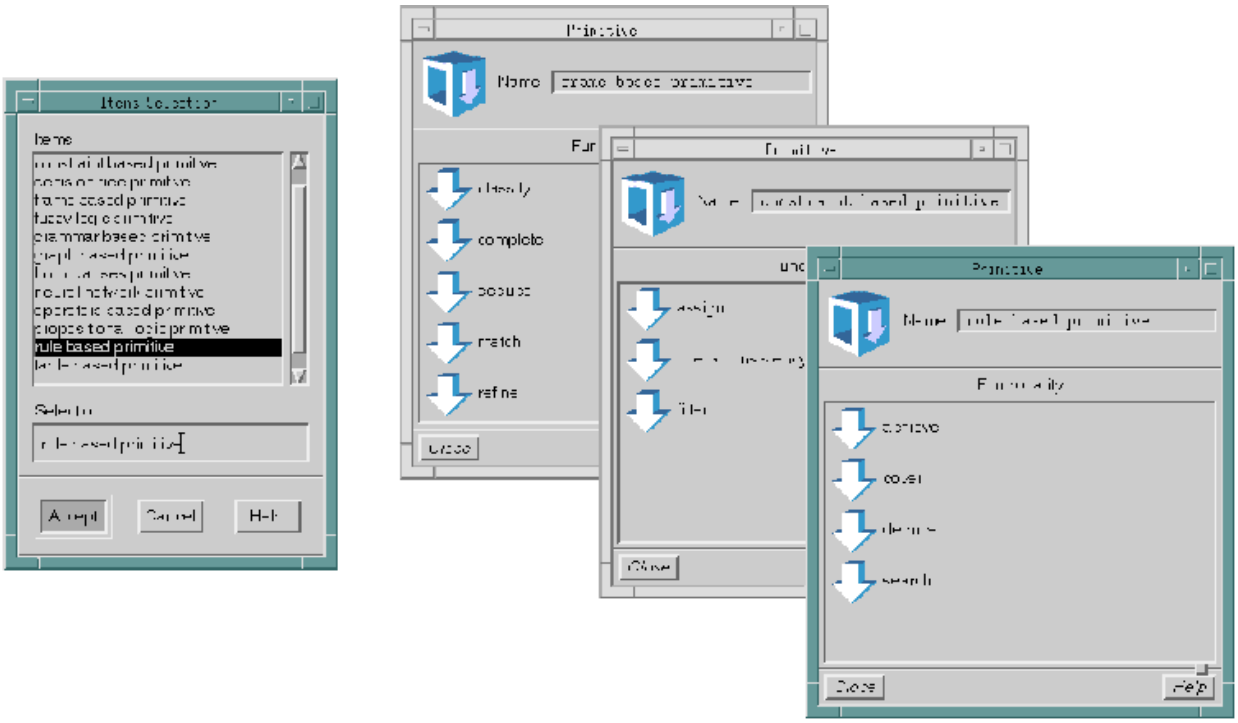

Figure 6: Examples of primtives of representation for operationalizing knowlede models.

On the other hand, the user can also create and edit conceptual vocabularies corresponding to the vocabulary perspective to define common terminologies to be used by different knowledge areas. From the main window (button generic model), and also from primary knowledge units, the user can create and edit vocabularies. Figure 5 shows an example of the definition of a conceptual vocabulary using the corresponding window. This window includes two text editors. One is for the generic view of the vocabulary (generic content) and the other is for the domain view (domain conent). When the user edits a generic vocabulary, only the generic editor is active. For instance, in this example, a generic vocabulary is presented showing only the generic content. This includes the definition of a set of concepts, using the Concel language, with clases-subclasses and attributes-facets-values. When the user creates a domain vocabulary, he/she can edit the domain content and can read (but cannot modify) the generic content. Typically, the domain content will include subclasses and instances of the classes defined at generic level. In the window of a vocabulary, the operator uses the buttons at the bottom level to validate the text, and adequate error messages are presented in additional windows when the text is not correct. KSM provides facilities in this window (and also in the window to edit methods) to load texts from a text file, or to save the vocabulary to a particular file.

In summary, following this procedure, the developer builds and edits a generic model with the three perspectives: knowledge areas, tasks and vocabularies. Then, the developer associates computable constructs to the model to produce an operational version of the model. In order to do that, KSM provides the library of primitives of representation that includes a set of basic problem-solving techniques.

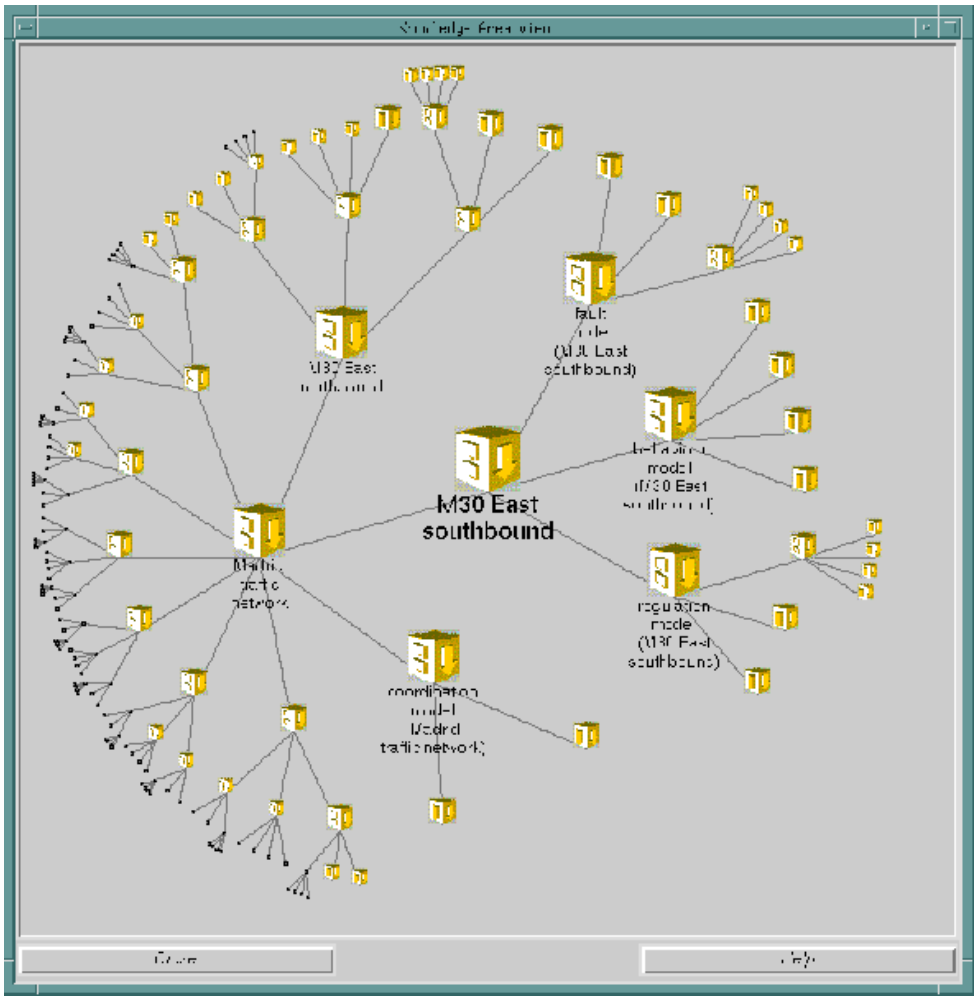

Figure 7: Example of the knowledge-area perspective of a domain model presented in a hyperbolic browser.

For instance, figure 6 shows an example of the components of such a library. On the left hand side, a menu shows an available set of primitives where each one follows a particular representation: constraints, decision trees, frames, fuzzy logic, grammars, graphs, horn clauses, neural networks, operators, propositional logic, rules, tables, etc. On the right side there are windows of three primitives, where it is presented the set of tasks that each primitive can provide. For instance, the constraint based primitive includes the tasks assign (to find an assignment of values for variables which is consistent with the constraints), check consistency (to check whether a particular assignment is consistent with the constraints) and filter (to reduce the initial set of values for each variable to a smaller set that is consistent with the constraints). Using this windows (and other more detailed ones that are not presented in the figure), the developer can select the most appropriate primitive to operationalize a particular primary knowledge area of the generic model. The primitives that the user of KSM can choose can be increased by programming new primitives of representation. This job must be carried out by a programmer in a previous stage of the model development. Once the new primitive is built and integrated in KSM, it will appear in the list of available primitives and the user will be able to use it to develop different applications.

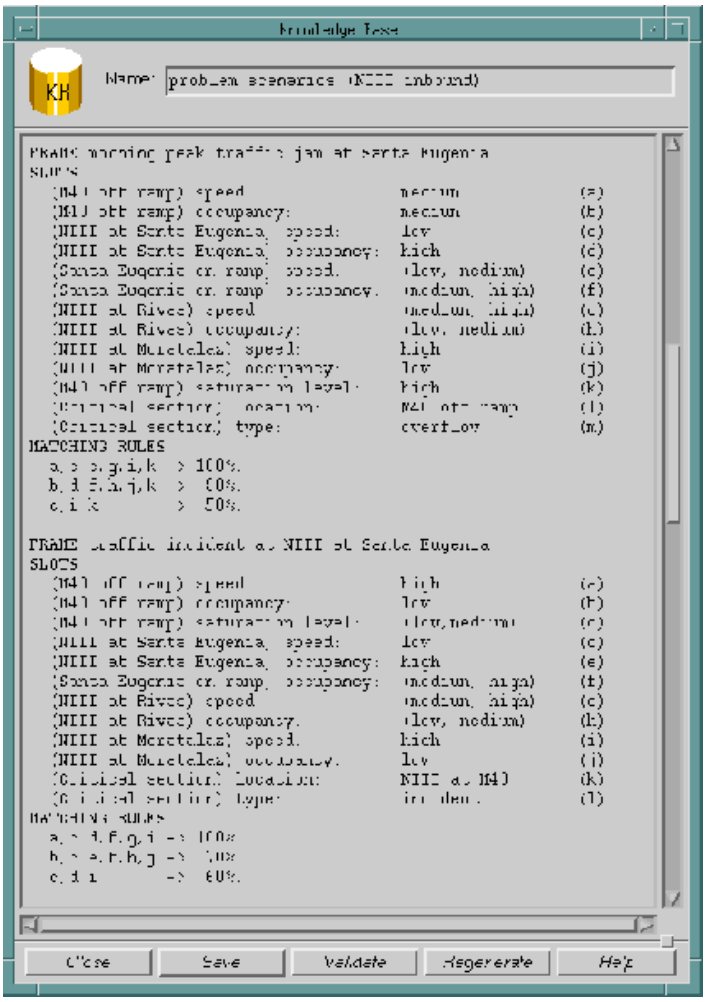

Figure 8: Example of knowledge base formulation in the domain of traffic control.

After the previous process, the generic model is ready to be used like a template for building domain knowledge models. To do that, the developer instantiates the pattern defined by the generic model into a particular domain by duplicating units or structures and adapting components (by filling in vocabularies and knowledge bases of primary areas). Typically the resulting domain knowledge model presents the same structure as the generic model but includes more components, given that several instances of the same generic knowledge area can be defined for a particular domain. The resulting structures of knowledge units at domain level are usually complex (with dozens of knowledge units). In order to provide a useful graphical view of such a structure and to simplify the access to particular units, KSM can present for the knowledge-area perspective what is called a *hyperbolic browser*. This browser shows the hierarchy projected on a spherical surface (with a kind of fisheye distortion) where the central node is viewed in more detail and the size of the units decreases as they go outwards. Given this perspective, the image provides a synthetic view of the complexity of the model. The hyperbolic browser initially displays the hierarchy with the root at the center, but the display can be transformed to bring other nodes (clicking them) into focus. In this way, the operator can go easily through diferent components of the structure in order to access particular elements and have, at the same time, a global view of the model. Figure 7 shows an example of a knowledge area view (using a hyperbolic browser) for a domain model in the field of traffic control of the city of Madrid.

Every knowledge-based primary unit of the domain model includes a particular knowledge base that follows the representation defined by the associated primitive to the primary unit. For instance, for the case of the problem scenarios primary unit of the model considered as example, a frame based primitive of representation was selected. Thus, when a particular copy of this unit is created for a domain, it is necessary to write its particular knowledge base. Figure 8 shows the window that presents KSM to allow the developer to write a knowledge base. This window is a text editor with procedures associated for validating and compiling, following the corresponding syntax of the representation language used by the primitive. In this case, the knowledge base is written using frames, where each frame represents a scenario of a traffic problem.

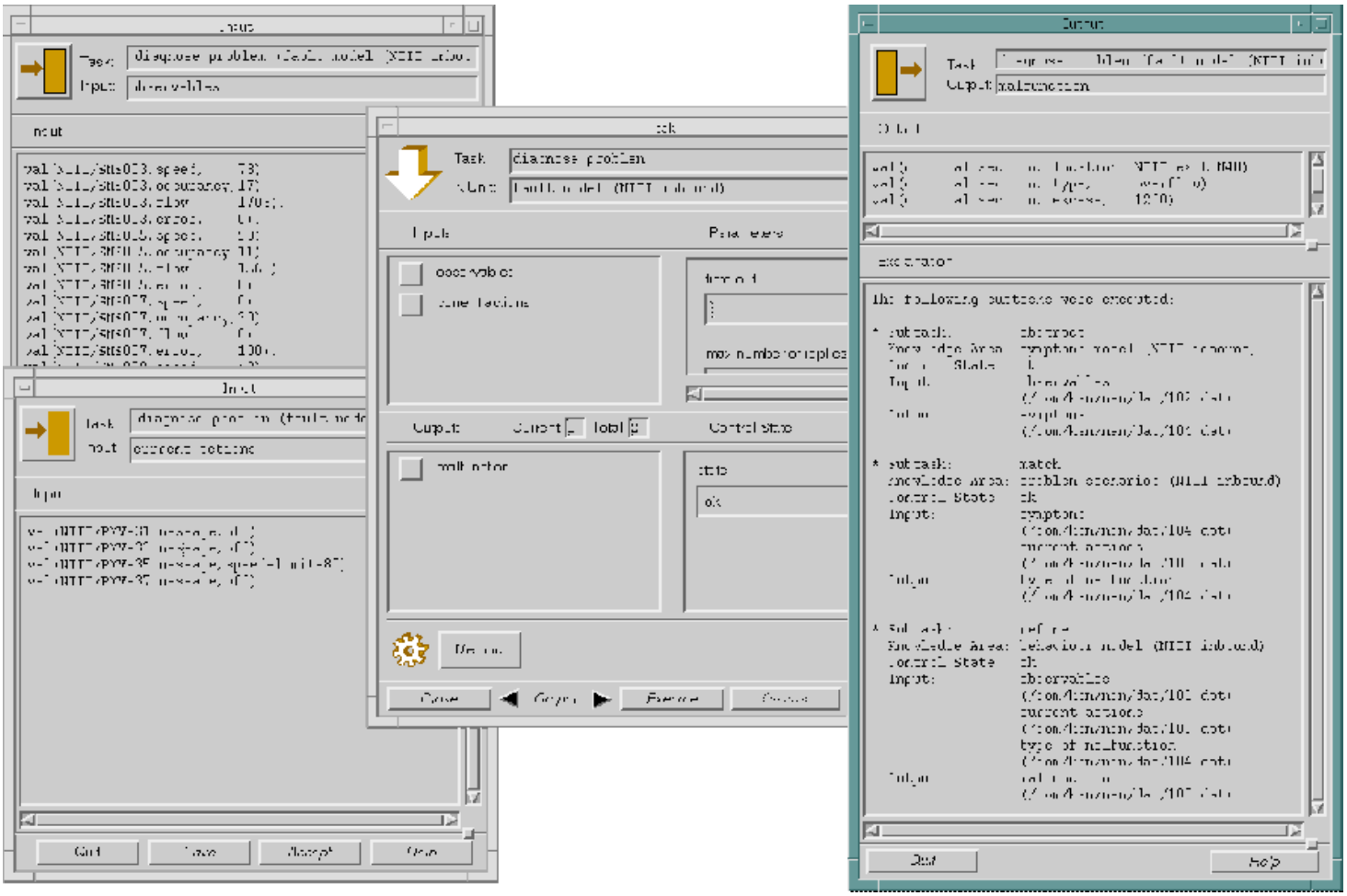

Figure 9: Example of task execution.

Finally, once the domain model has been created, it is ready to be executed. KSM allows the developer to execute tasks of knowledge units. When the user edits a task, the corresponding window includes inputs, and control parameters as it was shown in figure 3. The user can execute the task by doing the following procedure. First, the operator clicks one of the inputs of the tasks and a new window appears where the developer can write a list of input data. This can be done either by writing directly in the window or by loading an existing text file that contains this information. This has to be done for every input of the task. Then, the developer can choose certain modes of execution by writing certain control parameters (e.g. the maximun depth of the search tree, time out, etc.). Afterwards, the developer can execute the task. Once the taks has been executed, the developer can consult the result by clicking a particular output. The result is presented in an additional window that besides the results, presents an explanation about how the subtasks of the task were executed. For example, figure 9 presents an example of task execution for the case of the task diagnose problem in the domain of traffic control for a particular road called NIII inbound. On the left hand side there are two windows where the developer has written input data: observables, that contain values recorded by sensors about speed, flow, etc. of the traffic at certain locations, and current actions, that contain the state of variable message signs. In the center of the screen the task window allows the developer to execute the task using the button called *execute* located at the bottom level. The window on the right presents the output of the task, the malfunction. In this case, it is a traffic problem (overflow) at certain location with a certain level of severity. This window presents also an explanation showing the subtasks that were executed with references to the files that contain the information that was generated during the reasoning.

3. SUMMARY OF THE KSM ARCHITECTURE

The KSM environment includes, at the level of implementation, a set of software tools that support the operation with the user for building and executing knowledge models. These tools are:

- A library of reusable software modules (the primitives of representation), some of them highly reusable given that they are knowledge based.
- A user interface which presents a framework to develop the knowledge model following the KSM methodology. This interface consists of a graphical view window-based of knowledge components. Each entity (knowledge unit, vocabulary, etc.) is presented in a different window which associates a visual image. This interface provides graphical facilities to create, modify and delete knowledge units, primary units, conceptual vocabularies, tasks, methods and knowledge bases.
- The Link interpreter which allows to edit a text representing methods, verifies its syntax and interprets it during their execution.

- The Concel compiler which allows to define conceptual vocabularies and translates them into an internal structure verifying its syntax structure.

- A user interface to execute knowledge models. The execution can be done for the whole model or parts of it. Using this interface, the user selects tasks of knowledge units to be executed, provides input data and consults results and explanations.

The KSM runs on high performance Unix workstations of different firms (Hewlett Packard, Digital, Sun, etc.). The languages used for the implementation of KSM are C and Prolog. Both languages have been improved by adding object oriented features. Besides, X Window and OSF/Motif have been used to programme the user interface.

REFERENCES

[Alonso et al., 90] Alonso M., Cuena J., Molina M.: "SIRAH: An Architecture for Professional Intelligence". Proc. 9th European Conference on Artificial Intelligence ECAI 90. Pitman, 1990.

[Chandrasekaran, 86] Chandrasekaran, B.: "Generic Tasks in Knowledge Based Reasoning: High Level Building Blocks for Expert Systems Design". IEEE Expert, 1986.

[Cuena et al., 92] Cuena J., Molina M., Garrote L.: "Combining Simulation Models and Knowledge Bases for Real Time Flood Management". In "Computer Techniques and Applications", Blain W. y Cabrera E. (eds). Elsevier Applied Science. 1992.

[Cuena, Molina, 94] Cuena J., Molina M.: "KSM: An Environment for Knowledge Oriented Design of Applications Using Structured Knowledge Architectures" in "Applications and Impacts. Information Processing'94". Volume 2. K. Brunnstein y E. Raubold (eds). Elsevier Science B.V. (North-Holland) IFIP 1994.

[Cuena, Molina, 96] Cuena J., Molina M.: "KSM: An Environment for Design of Structured Knowledge Models". To be published as chapter of the book "Knowledge-Based Systems-Advanced Concepts, Techniques and Applications". Edited by Spyros G. Tzafestas. Publisher: World Scientific Publishing Company, 1996.

[Cuena et al. 96] Cuena J., Hernández J, Molina M.: "Knowledge Oriented Design of an Application for Real Time Traffic Management: The TRYS System". Proc. 12th European Conference on Artificial Intelligence ECAI 96. Budapest, Hungary, 1996.

[McIntyre, 93] McIntyre A.: "KREST User Manual 2.5". Vrije Universiteit Brussel, AI-lab. Bruselas, 1993.

[Newell, 82] Newell A.: "The Knowledge Level" In Artificial Intelligence Vol 18 pp. 87-127, 1982.

[Puerta et al., 93] Puerta A.R., Tu S.W., Musen M.A.: "Modeling Tasks with Mechanisms". International Journal of Intelligent Systems, Vol. 8, 1993.

[Steels, 90] Steels L.: "Components of Expertise" AI Magazine, Vol 11 (2), 29-49, 1990.

[Wielinga, Breuker, 86] Wielinga B., Breuker J.: "Models of Expertise". Proc. European Conference on Artificial Intelligence ECAI-86. 1986