An Extensible, Intelligent System for Applying Software Development Knowledge

by

Jennifer E. Grucza

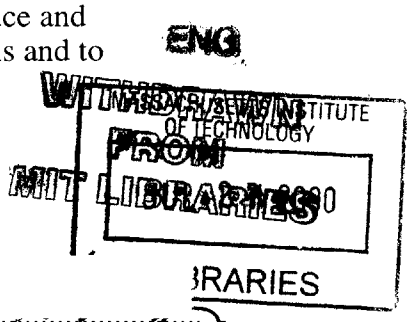
Submitted to the Department of Electrical Engineering and Computer Science in partial fulfillment of the requirements for the degrees of Bachelor of Computer Science and Engineering and Master of Engineering in Electrical Engineering and Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

August 27, 1999

[September 1999]



Author .....
Department of Electrical Engineering and Computer Science
August 27, 1999

Certified by .....
Howard Shrobe
Thesis Supervisor

Accepted by .....
Arthur C. Smith
Chairman, Department Committee on Graduate Theses

An Extensible, Intelligent System for Applying Software Development Knowledge

by

Jennifer Grucza

Submitted to the Department of Electrical Engineering and Computer Science

August 27, 1999

In partial fulfillment of the requirements for the degrees of Bachelor of Science in Computer Science and Engineering and Master of Engineering in Electrical Engineering and Computer Science

Abstract

Software development environments often suffer from ineffective collaborative collaboration, with the result that projects run over schedule, necessary tasks aren't completed, designs aren't as elegant or robust as they could be, and so on. This document describes an implementation of a system which provides a framework for storing and handling knowledge about software development environments, with a specific emphasis on handling bug reports intelligently through the use of a rule-based system.

Thesis Supervisor: Howard Shrobe
Title: Associate Director, MIT Artificial Intelligence Laboratory

Acknowledgements

I'm grateful to Howard Shrobe for being such a helpful and understanding advisor. I'd also like to thank my future employers at i2 Technologies for their patience in waiting for me to finish this thesis! Finally, for their patience, love, and support, I thank my parents, Leo and Susan Grucza, as well as the rest of my family.

This paper describes research done at the Artificial Intelligence Laboratory of the Massachusetts Institute of Technology. Support for the MIT Artificial Intelligence Laboratory's artificial intelligence research is provided in part by the Defence Advanced Research Projects Agency of the Department of Defense under contract number F30602-97-2-0013.

Contents

List of Figures

1 Introduction

1.1 Motivation and problem statement

Probably the most important factor in the success of any project, in any field, is the efficiency of communication between project participants. In a small group environment, handling communication is a relatively simple task. However, as the numbers involved in a project increase, the communication overhead required for effective collaboration increases dramatically. This thesis deals with improving communication and collaboration in the domain of computer software development, an area which certainly suffers from its share of impediments to effective communication. In addition, the software industry is a perfect domain to coordinate using online tools, since online tools (email, the web, newsgroups, etc.) are already being used to a great extent in collaboration. In addition, the actual work being done is also on the computer, which makes it possible to *directly* link messages or annotations to their subject matter [13].

Thus, our goal is to build a system which embodies the knowledge of software development environments in the general sense, as well as knowledge of the specific environment at hand, and furthermore, to use that knowledge to help guide the process of software development, whether it be in reporting and tracking bugs, aiding product design, critiquing pieces of code, or streamlining the development process in some other manner.

In referring to knowledge of software development environments in the general sense, we include such information as what types of users are generally present in such an environment, general skill-sets or subject areas involved (such as various programming languages, interpersonal skills, project management skills, ability to learn quickly or handle a wide range of situations, etc.), stages of the software development life-cycle (brainstorming, requirements analysis, specification, design, implementation, testing, documentation, deployment, and maintenance), as well as other general knowledge about

software development. But most importantly, we must know the relationships between these pieces of information. For example, it would be important to realize that implementation follows design, and not the other way around. Or that project managers must have knowledge about the project they are managing, and that they have a managerial relationship to other users in the system. These types of relationships are what give this web of knowledge its real power.

In addition to this general knowledge about software development environments, it is necessary to represent knowledge specific to the users' environment, such as information about users themselves (names, email addresses, skills, interests, managerial relationships, responsibilities, etc.), information pertaining to the particular software being developed, knowledge about various groups working within the company, and other similar information. In addition, it is desirable to allow modification of the general knowledge built into the system, as it is impossible to make a model with any useful amount of detail which fits all individual organizations perfectly.

Built upon this large knowledge base, we can then add any number of useful applications which make intelligent use of the information it provides, inferencing by way of a classic rule-based system (with various general rules provided, and more which can be added by the user) for greater effect.

The benefits to be offered by such an online system are many. First of all, such a system would prove to be a time-saver. By automatically retrieving needed information, or by figuring out which person is best suited for any particular task, one saves time which would have otherwise been spent searching for information, trying to find someone with the right expertise, or investing resources in a person not ideally suited for the task. Time and energy are saved by finding the right resources, at the right time, so that the work doesn't have to be redone or modified later.

This thesis seeks to provide such a system - a tool for aiding collaboration and easing software development. Specifically, we will be focusing on the domain of bug tracking.

1.2 Background

Considerable research has gone into various aspects of the issues raised in the introduction, from the general management of software development environments to various tools used to manage communication in an environment with many collaborators. Though not all of the following is directly applicable to this thesis project, it is of interest, being closely related.

1.2.1 *Software Development Management*

Managing a software development effort involves many factors. One must coordinate the efforts of many people doing various different tasks in various roles, the interrelation of the various components of a system, and the schedule by which development should take place.

In any software project, there will be a great number of people filling different roles related to the project. The obvious ones are the roles of software engineers, software project managers, documentation writers, quality-assurance engineers, and the users of the system being developed. In addition, other roles that can be involved include system analysts, maintainers, user-specialist liaisons, user department managers, vendor representatives, contract monitors, customers (in the case where the users aren't the direct customers of the system), along with others. Coordinating the efforts of people in such disparate roles is a very complex problem. In addition to the widely different roles and occupations involved are the different ideas held by each person as to what work needs to be done when. If the participants are not managed effectively, bottlenecks occur and important pieces of work which may have been expected to get done, but were not explicitly specified, may not get done when necessary, or at all. These problems can

severely harm the schedule to which the development effort is tied, resulting in serious delays and sometimes even cancellation of projects.

Other important factors in the management of software development (besides the roles and occupations filled by participants) are the set of current tasks being performed by each person (which is not necessarily constant even though the role or occupation may stay the same), the skill level and skill set of the participants (influencing the speed at which tasks will get done by a participant and which tasks are assignable to any particular person), the relationships of the worker to other people around him/her (e.g. hierarchical management chains, project leaders, fellow project developers, etc.), and turnover of employees. This last is especially relevant in the software industry today, as people work for a couple of years and move on to a different job within the company or a different company entirely. They take their expertise with them, often making it very hard for the next developer to maintain their code.

These many and varied problems have been dealt with in industry in a variety of different ways. Management of scheduling and of the assignment of tasks, among other things, are simplified by abstracting the software development process into a life-cycle with distinct stages. Management of communication has been addressed by various tools besides the traditional person-to-person meeting.

1.2.2 *Tools for Managing Collaborative Communication*

One concern in improving collaborative communication is being able to record discussions about various development issues so that they may be reviewed later. This can be useful in reviewing the reasons behind design decisions, reviewing possible alternative implementations or designs that were rejected, and as a general means of documenting the flow of information between people involved in the production of a software system.

Another concern is directing discussion in an organized and efficient manner, where arguments are presented clearly.

A good model for handling these issues is the issue-based information system (IBIS). This model has been applied to a wide array of applications, from law to scientific collaboration, to software development. The IBIS model is based on the laws of argumentation: issues, or problems, are the focus of the discourse. Positions, or suggested solutions, can be made on an issue, and arguments can support or argue against a position. These three types are represented as nodes which can be connected by links. A link from one issue to another could be of the type "generalizes" or "specializes," for instance, and a link from an argument to a position could be either "support" or "object."

The general framework of IBIS can be extended and specialized to fit any particular domain. In a software development environment, issues could be questions as to how to design a particular part of a system. Positions would suggest possible designs, and arguments would comment on these positions. Any of these node types can be specialized if needed. Instead of having one generic argument type, there could be sub-types for support, objection, justification, request for clarity, etc.

The feature of using such a system is the structure it introduces into discussion. It can capture just as much knowledge as any other sort of discussion medium, but with the advantage of being organized from the outset. There is no need to go over notes and organize things at a later date, because in order to enter new nodes, you have to figure out what part of the discussion you want to add to. In addition, this construction allows a person to go back later and see a well-organized graph of the discussion, with issues raised and ways they were resolved. There is no need for additional documentation. Retrieval of information is easy and convenient.

2 A Long Term Vision

2.1 Knowledge sharing

In a perfect world, all the administrative (and other) software packages in use by an organization would be linked together, harmoniously sharing their knowledge and working together to smooth the running of the company. Such a system would draw upon this wide range of knowledge to help users of the system in various tasks, reducing the grunge-work done by users, as well as taking on an advising role in certain circumstances where the user needed assistance, or simply to provide the user with alternatives that might have otherwise been passed over.

Imagine that the software used to keep personnel records (including simple name/ address information as well as performance reviews and information about user skills and interests) were linked to scheduling and planning software, software development tools (such as integrated development environments (IDEs)), inventory information (in other words, what resources are owned and used by the organization - computers, telephone system, printers, etc.), and finally, that each piece of software in use maintained information about itself which was readable by the rest of the system. Figure 2.1 shows a problem object model [8] which describes what information an example system might have to keep track of. As you can see, there are many interrelationships between very different kinds of data. Using an automatically interconnected system, such as described above, has the opportunity to free up a significant amount of manpower, which is currently wasted in transferring information from one medium to another or duplicating it in more than one medium.

In addition to simply saving users from needing to duplicate information or transfer information back and forth, having such a large interconnected system would allow other applications to be built on top - applications which could take advantage of this network of

knowledge to aid users in less well-defined problems. The following sections describe some of these possible applications.

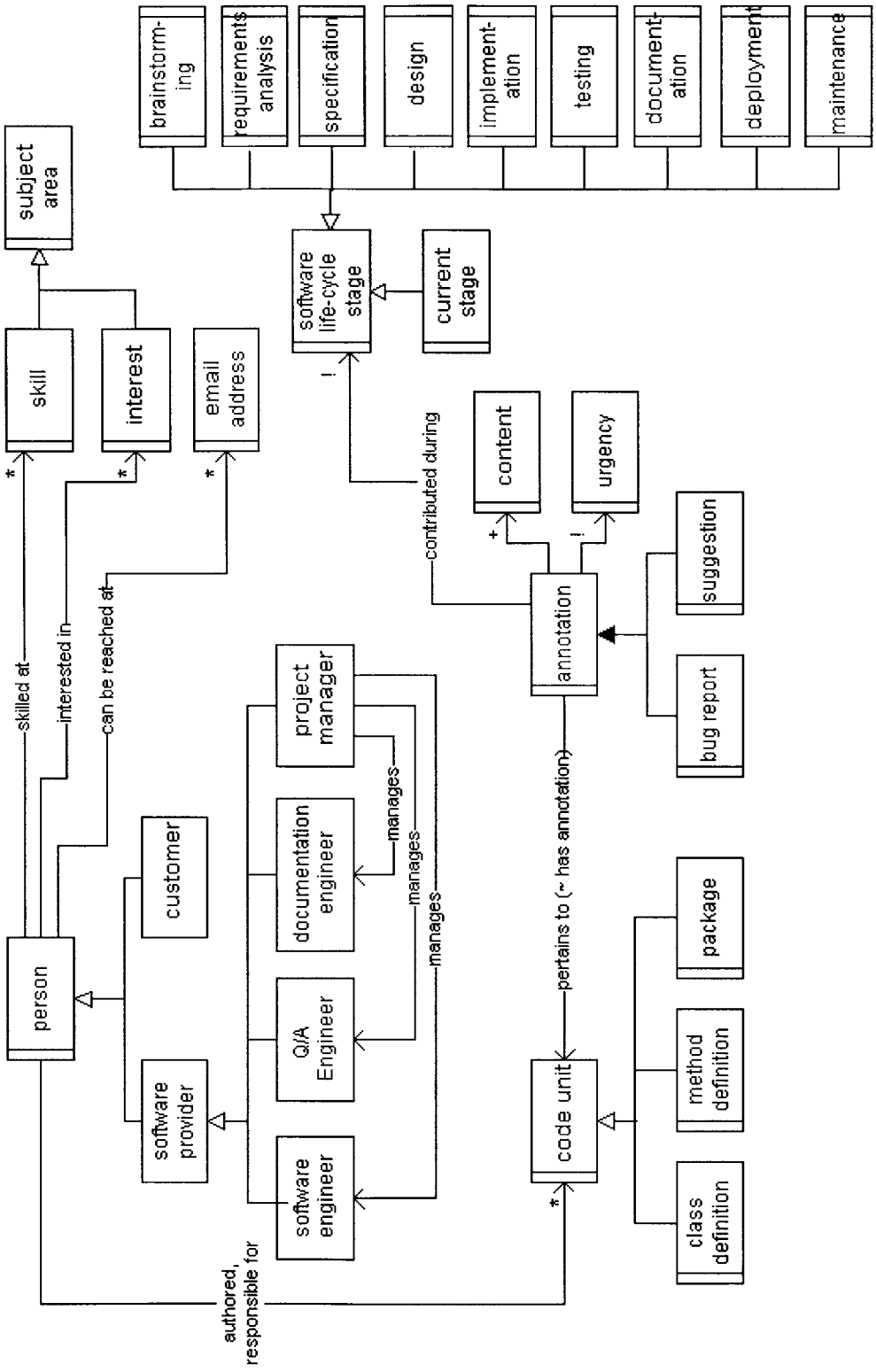


Figure 2.1: Problem Object Model

2.2 Possible applications

2.2.1 *Code unit annotation*

In section 1, we briefly mentioned that, since software development is done on computer, it should be possible to directly link this work - the actual code being written - to the rest of the system. Christopher Vincent, in his master's thesis [13], devised a system to do this very thing. His system, DATE, provides a framework by which individual units of code may be annotated and discussed in an IBIS-like manner. This system, and future work done along the same path, is a perfect example of the kind of application which would benefit from the wide interconnection of administrative and developmental applications (as well as vice versa). Notably, it would fit in quite well with the bug reporting and regression test applications mentioned below.

2.2.2 *Automatic creation of module dependency diagrams*

A common problem that occurs when a large group of people works together on a piece of software is for small changes in one module to cause unexpected, unwanted effects in other modules. Sometimes these unexpected effects don't show up immediately, and when they do, trying to find their source is like finding a needle in a haystack. If dependencies were mapped out automatically, so that any time a programmer went to modify a module, a list of dependant modules appeared on his screen, many of these unexpected problems would be prevented. The programmer would be able to examine the dependent modules to determine the effects of his changes, and thereby fix any resultant problems. Of course, this feature wouldn't be quite as important if, for each module, its requirements, effects, and side effects were comprehensively and accurately documented. Unfortunately, in the real world this does not necessarily happen consistently. Even when all modules are documented, the wording can be ambiguous or difficult to understand. Additionally, it is sometimes the case that the behavior, and thus the specifications, of a module must be altered, rendering the provided documentation out of date.

Work has already been done on automatically generating such dependency charts. (See [10] for many examples of such.) Incorporating such tools into our interconnected system would further allow smarter testing, bug tracking, design of new versions, etc.

2.2.3 *Assisting the design process*

Not a very well-defined problem, but an interesting one, is trying to assist software developers in the design process. General knowledge about software design could be entered into the system, as well as previously considered or implemented designs, with knowledge about what their strong and weak points were. One major problem with an application such as this is trying to codify all the applicable knowledge. What knowledge representation would work best? We say "assist" in the design process - how would this assisting take place? Perhaps proposed module dependency diagrams could be fed into the system, which would then critique the proposed designs based upon previous experience.

2.2.4 *Reporting and tracking bug reports*

A more concrete and tractable domain is that of bug reports. Consider the situation where a company's main product is currently on the market in several versions, with another, new version currently in production. If a bug report were to come in, how would we know

- its importance
- who should know about it
- when it should be fixed
- how it should be fixed
- whether it should even be fixed at all?

We could ask the reporter of the bug to characterize the urgency of the bug for us, and this would help us determine how important it is to fix the bug. But what happens when you get the novice computer user frantically swamping your system with numerous, supposedly very urgent bug reports which don't report bugs after all, but are actually only

misunderstandings caused by the user's unfamiliarity with the idiom or the fact that he did not even open the user's manual or help files? It would be beneficial if we could somehow recognize these sorts of reports and give them a lower priority, or assign fewer people to deal with them (or assign someone with nothing better to do). Conversely, if we receive a bug report from a major customer, with great clout and buying power, we want to do our best to resolve their problem as quickly and satisfactorily as possible. By incorporating our bug tracking tools into a large interconnected system, we could take advantage of this sort of knowledge stored elsewhere in the system, and flag the report as being very important. Then the fastest minds could be put to work to solve the problem. We would ask a larger number of people to be involved in the resulting discussions, and make sure we have people who are knowledgeable about the subject areas involved.

Which brings us to the next question: how do we know who is best able to resolve any given bug? If our system stores information such as people's interests, skills, work histories and the like, all we need is a way to match up these attributes with what we know about the bug. Section 3 describes an implementation of this approach, which uses a rule-based system for the reasoning.

In addition to knowing whether a bug is important, and knowing who should be involved in fixing it, it would be nice if we knew if it was better to address the bug immediately, or wait until a later time. For example, a company might receive a bug report while they were finishing up testing and documenting their product before shipping. This is probably not the time to be focusing on bugs which would require major design overhauls. If a quick but dirty fix were possible, we might wish to let those responsible for fixing last minute bugs know about the report. Either way, we would probably want to make sure the bug report also gets addressed at a slightly later time in the software life-cycle (actually, to be more precise, in the *next* cycle, or rather, by the people working on

the next version of the program). In this case, we would want it to be seen and discussed by a larger audience - by those responsible for larger scale design as well as those responsible for the lower-level implementation or testing.

In an ideal environment, the system would even be able to help developers figure out how to fix the bug. Perhaps it would have some way of cataloging bugs or finding previously solved bugs which match the current bug's profile closely. This might be based on more of a frame-based view of knowledge representation rather than rule-based.

2.2.5 *Running regression test suites*

Running regression test suites automatically, and determining what to do when tests fail, is a problem very similar to the bug reporting problem described above, except for the fact that we are not dealing with another human or organization at the other end. Furthermore, with regression tests, one often gets repeat failures of the same bug. It might be desirable to treat these repetitions differently than the initial failure notifications, whether that be by ignoring them or simply narrowing the number of recipients or something else. The system could also cross-reference bug reports and test case failures to look for correspondences.

2.2.6 *Hiring new personnel*

An application not specific to software development environments is that of hiring new personnel. Such a system as we've been discussing could be used to advise even in this area, based on what skills are already covered in the organization and what skills the potential hires have, and even perhaps incorporating knowledge of interpersonal relations. (For example, does Joe, the future manager of whoever gets hired, work well with everyone or only with certain people? Which candidates have traits compatible with those held by members who are already present in the group they would be hired into?)

2.2.7 *Calendar application*

Another general application of our interconnected system is that of scheduling and event tracking. People's individual calendars could automatically be updated with meetings scheduled from a central project planning application, since the system would have information about which people work in which groups (and even who would be on vacation, and couldn't attend any meetings).

Above are just a few of the possible applications possible in a system where all applications share information and a repository of rule-based knowledge for reasoning about the information.

3 Project Design and Implementation

3.1 Overview

The system implemented for this thesis was by necessity of smaller scale than a real system for use in a commercial environment would have to be (much less the ideal system described in the previous section). Thus the focus was narrowed to a manageable size. Specifically, the application chosen for the project (in addition to providing a general storehouse of knowledge) was a bug tracking system. Though the focus was narrowed, the rest of the system was designed to be as general and extensible as possible.

Figure 3.1 outlines the module dependency diagram (MDD) [8] that describes this system (dubbed "DevelopmentConsultant"). The program is divided into three main packages: the set of core classes, which implement the framework for creating and modifying facts about the software development environment; the rules package, which handles knowledge about how these facts relate to each other; and the set of graphical user interface (GUI) classes, which tie the application together, manipulating the other packages and providing the bug tracking application. The following segments will describe these sections of the program in more detail.

If you examine the MDD, you will see that nowhere is there mention of any classes to represent such things as users, subject areas, stages of software development, programs, modules, and other such obvious objects that occur in any software development environment. Indeed that would be a valid way of organizing such a system, and one that was considered by the author. However, explicitly building such modules into the system would have the effect of restricting user customization. The program design would have to incorporate any possible type of object which might be used in a software development environment in order for the program to be usable by more than a few organizations. No organization would use a system with regularity if it didn't account for certain important

characteristics used by that organization. And unfortunately, the set of software development environments, or any other set of environments, is not nicely homogenous. Each organization may favor different ways of doing things, which would mean a system with such objects hard-coded into the program would not get used by the majority of organizations.

However, if we instead have a way to make our system easily extensible for the user, able to add new kinds of objects without delving into the code at all, it will be more likely to be useful for a larger number of organizations.

That concept is behind the design of this system. By bypassing the initial impulse to explicitly design the software knowledge into the system itself, we free the system to be used by any group - in fact, the core framework in this system could be used by any group, not only software development environments! The way this has been accomplished is to introduce some of the concepts of object-oriented programming in a somewhat restricted manner. Briefly, the design is made up of a group of Factories, each of which contains Types, which can be thought of as sub-types of each Factory, as well as Instances of the Factory, which can "inherit" properties from various Types in the Factory. This will be discussed in more detail later. Suffice it to say, by introducing this structure, we can mimic the hard-coded design, yet retain easy extensibility. Instead of hard-coding in a class for users, with subclasses for programmers, managers, documentation writers, etc., and creating then instances of these classes - we can simulate similar behavior through the use of these Factories, Types, and Instances, which can be changed on the fly through the user interface.

However, with these advantages come disadvantages as well. The user is given much freedom to modify, add, and remove new Factories, Types, and Instances, but this can sometimes mean problems for other parts of the system. If the bug reporting application

depends on the fact that there is a Factory called "Urgency Descriptions" with Instances named "severely urgent, progress impossible", "moderate urgency", and so on, and the user removes one of these Instances, the bug reporting application might not perform exactly as intended. The program would not break, however, the user would be deprived of the full functionality otherwise available.

Is there a solution to this problem? Perhaps. One could provide warning messages if the user tried to modify or remove elements which are used by other parts of the system. This would not be too difficult a task generally. One could include a flag on any object which could be set to true when a second object indicates that it depends on the first to function properly. But this situation gets hairier when one takes into consideration the rules in the rule-based system, which also depend upon certain objects' existence. The problem is perhaps still solvable, but the question is, how would the design of the program have to be distorted to provide such a means, and would the complexity be worth it? Alternatively, the simple solution is just to new allow users to remove any object, but this is also not satisfactory.

Thus we have the arguments for and against the outlined design. As will be discussed again later, if the accessibility for changing Types and rules were restricted to knowledgeable personnel, this problem would not be terribly significant, and such a restriction is probably desirable anyway.

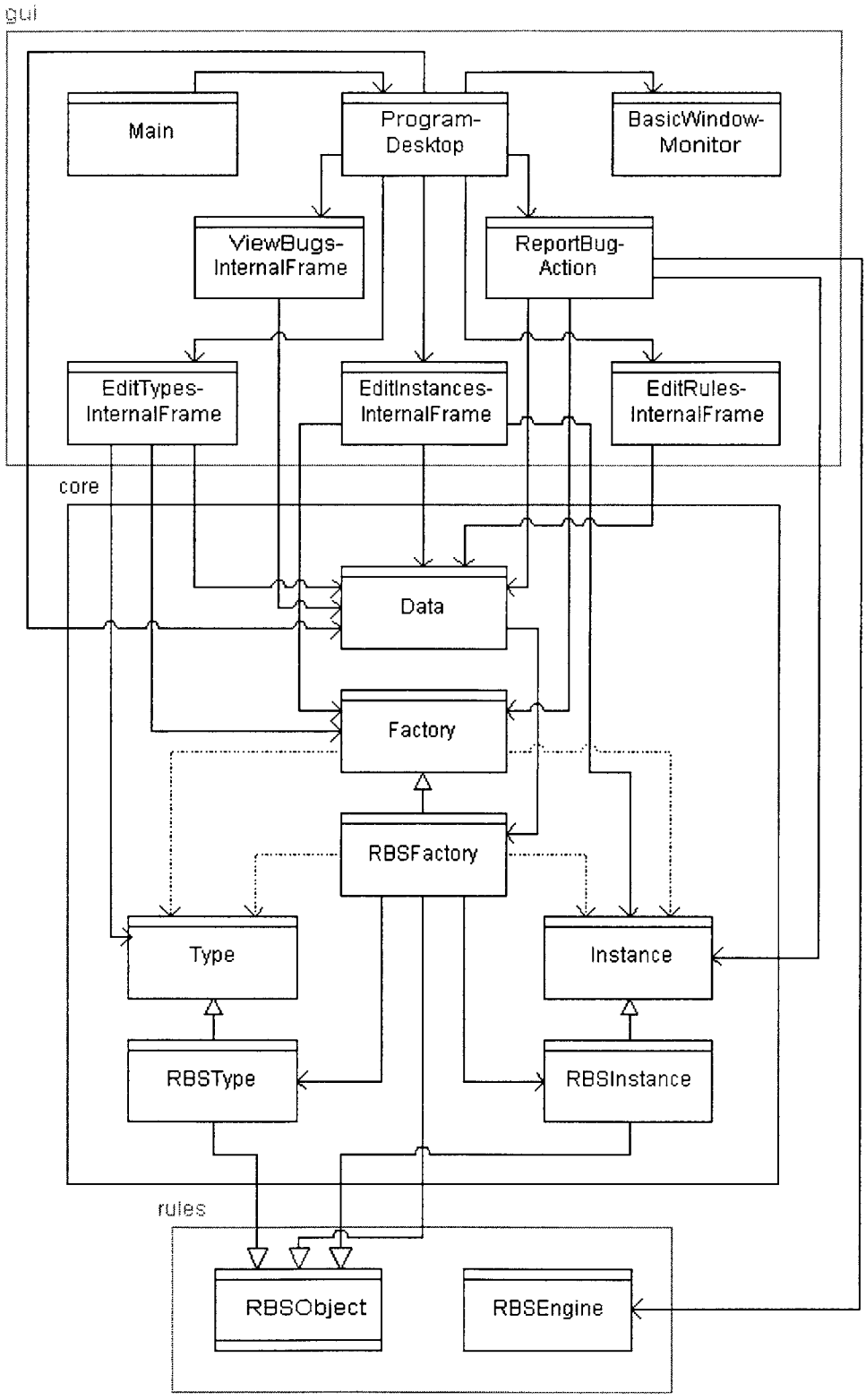


Figure 3.1: Program Module Dependency Diagram

3.2 Core classes

The core set of classes which comprise this program include Data, Factory, Type, Instance, and, additionally, RBSFactory, RBSType, and RBSInstance. The function of this set of classes is to hold all of the knowledge in the system - in this case, software development environment knowledge, though these classes could be used to hold any sort of knowledge desired. The highest-level class functionally, Data, serves as the main repository of this information. The remaining (non-RBS) classes hold specific parts of this knowledge base. We will describe this framework by working from the specific to the general.

An Instance object represents an object in the environment, such as "John Doe", "Common Lisp", "Requirements Analysis", and so on. As you can see, an Instance doesn't have to be a concrete object - it can be a more abstract idea. Each Instance object has one or more Types to which it "belongs," and inherits property fields from.

A Type represents a group of Instances which share common properties. For example, the Instance "John Doe" maybe belong to the Types "programmer" and "manager". Likewise, "Requirements Analysis" belongs to the Type "waterfall model" (stage). Each Type keeps track of what properties it has (such as first & last name, email address, etc. for various user Types). If an Instance belongs to more than one Type, it has the properties of all of them. Note: properties only have values in relation to Instances. Types simply outline what property names exist, as well as whether the value for each property is a String, or can be chosen from one of the Factories, but they have no property values themselves.

A Factory is, in essence, simply a collection of similar Types, as well as all the Instances falling under those Types. (An Instance can only "belong" to Types within one Factory.) To continue our earlier example, we would have a "Users" Factory, with Types

such as "programmer", "manager", "customer", etc., and Instances such as "John Doe", "Mary Smith", etc.

The Data class then stores a collection of Factories (in this implementation, Factories have been included for users, subject areas, software life-cycle stages, machine platforms, operating systems, programs, program modules, urgency descriptions (for bug reports, for example), and bug symptoms), with each Factory containing its own specific Types and Instances. In addition to this, the Data object keeps track of rules and bug reports (see later sections).

The final classes in the core package are RBSFactory, RBSType, and RBSInstance. These are simply subclasses of Factory, Type, and Instance which have been extended for use in the rule-based system. This will be discussed in section 3.3.

3.2.1 *User interface for entering Type and Instance information*

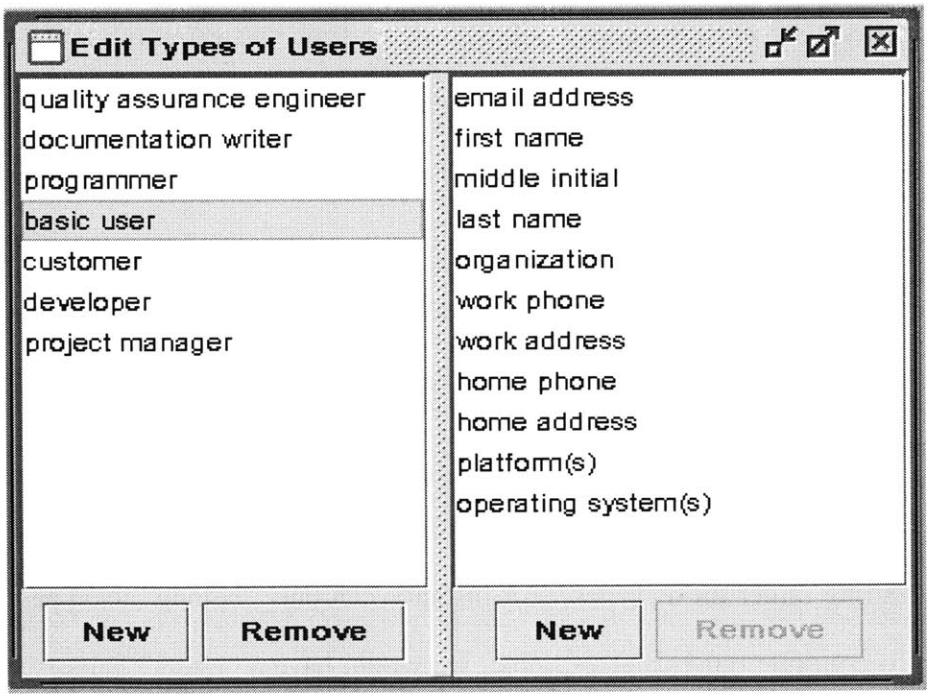


Figure 3.2: Editing Types & their properties

Figure 3.2 shows a screenshot of the user interface for creating new Types. In this case, we are editing the Types found in the "Users" Factory. The interface is relatively intuitive, with the left panel allowing users to view, add, and remove the Types, and the right panel displaying which properties are available for each Type. (Notice how the Remove button on the right is disabled - you cannot delete a property if none are selected.) If the user tries to add a new Type or a new property when one already exists by the same name, a dialog box is displayed, warning the user and preventing her from accidentally overwriting an existing Type or property. (However, two different Types may have both have a property with the same name.)

Once the user is satisfied with her choices, she can then create new Instances of the Types which she has specified. Figure 3.3 shows the window used for editing Instances. In this example, we are looking at the information entered for the Instance "Grucza, Jennifer", which is an Instance of the "Users" Factory. The top left hand panel lists all the Instances in this Factory. The top right hand panel lists the Types which the selected Instance belong to. These types can either be specified when a new Instance is created, or at a later time, using the New and Remove buttons in the top right panel.

The bottom panel is devoted to the property values held by the selected Instance. The list of properties on the left consists of the combined lists of properties specified for the Types the Instance belongs to, which we specified earlier. There arises the question of two different Types having the same property name, however. In the case where all Types are specified upon creation of the Instance, we can simply ignore the fact that a property name occurred in two Types. There are no values assigned yet, so this causes no conflict of values. In the case where we later add a Type to an already existing Instance, and there is a conflict with the new Type having the same property name as one of the Instance's existing properties, we must make sure to keep the value we have already stored for the

Instance, and not overwrite it with a null value, which is what would normally happen when we add a Type with a new property. Once we have made this adjustment, we are fine.

To reiterate, the bottom left panel holds all the property names for the selected Instance. When a user selects one of these properties, the bottom right panel then allows the user to enter a value or values for that property. Since there is more than one type of value a property may have, we have different interfaces for entering them. The one pictured in figure 3.3 is for when we have specified that a property value is to be chosen from the Instances of one of the Factories. This allows the user to choose one or more values from a range of Instances from within a Factory. In this example, we are choosing what values to include for the property "moderately skilled at". The values are being chosen from the "Subject Area" Factory, which we have filled with Instances such as various programming languages, software design concepts, work skills, etc. The user may add and remove as many Instances as she wishes. Then to confirm, the Enter button must be pressed. The second type of property value is a simple String, which the user enters into a field, after which they must press the Enter button to enter the value into the database (not shown).

Now that we have discussed the framework of Types, Instances and the rest of the core classes, as well as the graphical user interface used to manipulate their values, we will discuss how we can use the data stored in this framework for more interesting and useful purposes.

Figure 3.3: Editing Instances: their parent Types and property values

3.3 Rule based system

One of the classic techniques in Artificial Intelligence for reasoning about data is the *Rule-Based System*. In such a system, there is a collection of *rules*, each of which is made up of a left-hand side (LHS) and right-hand side (RHS). A rule's LHS contains patterns which are then matched against a set of *facts*. When a complete match is found according to the rule's LHS, the RHS specifies an action or actions to be performed, such as asserting a new fact, which can then trigger other rules to fire in turn. In this manner, we can begin to approximate human knowledge and reasoning as a group of logical steps which help us reach a new understanding of the state of a system. (E.g. "I need help figuring out what this piece of code is supposed to do." Well, who wrote the code? "Joe is the person who wrote that code." Can you ask him about it? "Joe has left the company." Then, who else would it make sense to ask? "Joe's manager for that project was Bill." "Managers generally have knowledge about the work of those they manage." Therefore, we can ask Bill. However, we can also proceed along a parallel path in order to get more information: What topics does the code cover? "The code involves the TCP/IP protocol." Who is knowledgeable about that subject? "Mary and Bob know a lot about network programming" and "TCP/IP is used in network programming". Therefore, we can ask Mary and Bob in addition to Bill.)

The rule-based system implementation used in this project is the Java Expert System Shell (jess) [6]. This implementation is based on the widely-used CLIPS system produced by NASA [7], which was in turn derived from OPS-5. This implementation was chosen mainly to ease integration with the rest of the program, since it is compatible with Java.

The RBSEngine class in DevelopmentConsultant's rules package is an extension of jess's Rete class, which makes up the rule engine itself. Each Rete object has its own knowledge base, rules, agenda, global environment, and so on. The RBSEngine class

extends Rete by adding deftemplates for facts used by RBSFactory, RBSType, and RBSInstance, as well as providing a mechanism for loading in the set of facts from a Data object, which represents the knowledge of the system. RBSFactory, RBSType, and RBSInstance use the defined templates to create these facts.

The other class in the rules package is the RBSObject class. This is simply an interface which is implemented by RBSFactory, RBSType, and RBSInstance. It specifies the behavior for any object which is to be used by the rule system - namely, converting the information it contains into fact form, which can then be used by the rule engine.

3.3.1 *User interface for editing rules.*

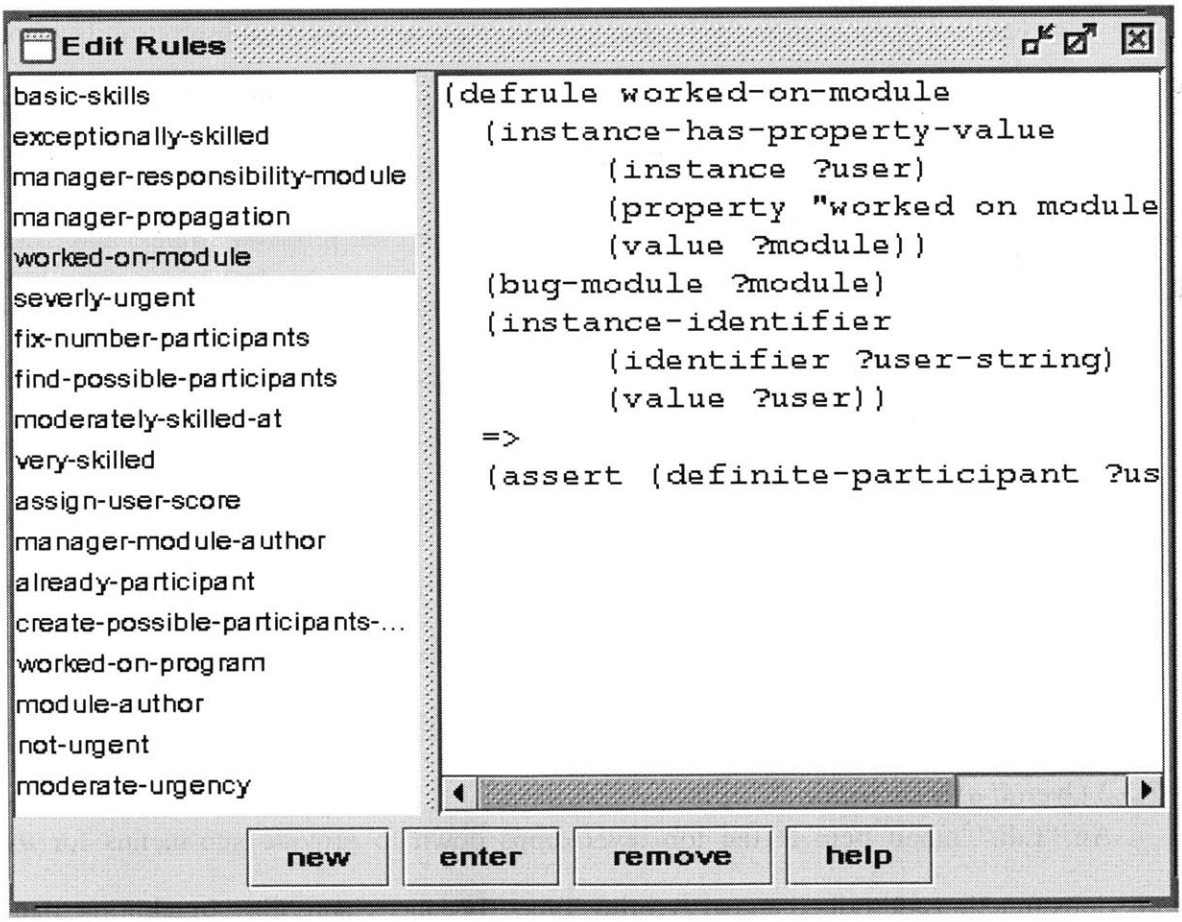


Figure 3.4: Editing rules

Figure 3.4 displays the window used in editing rules. When a new rule is created, the user is asked to provide a name for the rule. The new rule-name appears in the left panel, and when that rule is selected, the right panel shows a sample rule layout, with the rule name already in place, as follows:

```
(defrule new-rule-name
  (                    )
  =>
  (assert (            )))
```

The user then may fill in the skeleton rule or modify it as he or she chooses. The enter button must then be pressed for the new definition to be entered into the system.

As you can see, this user interface requires that the user have knowledge of the CLIPS/ jess rule syntax in order to effectively change or create rules. This does limit usability, and in a real system, it might be beneficial to separate the editing of Types and rules so that only selected personnel (such as systems administrators) have access, limiting general use to editing Instances (e.g. creating new users and filling in their property values) and using the bug tracking or other future applications. Although this interface has the disadvantage of requiring knowledge of CLIPS/jess, it does have the advantage of allowing greater flexibility in the behavior of the rule-based system than would otherwise be available. An alternative interface might have been more restrictive, letting the user fill in only selected fields in a rule template and asserting certain facts (which could be chosen from a menu or somesuch). This could significantly cut down in the usefulness of any application built on the rule system, though. So either way there are advantages and disadvantages.

3.3.2 *Overall user interface*

An "Edit" menu item at the top level drops down to provide sub-menus for the previous functions described above - editing Types, Instances, and rules. In addition, there is a "File" menu used for saving and loading state (the user is asked to confirm since each

of these operations has the possibility of accidentally erasing data) as well as exiting the application. The third top-level menu, "Bug Tracking," provides the interface for the user to choose the applications described in the following section.

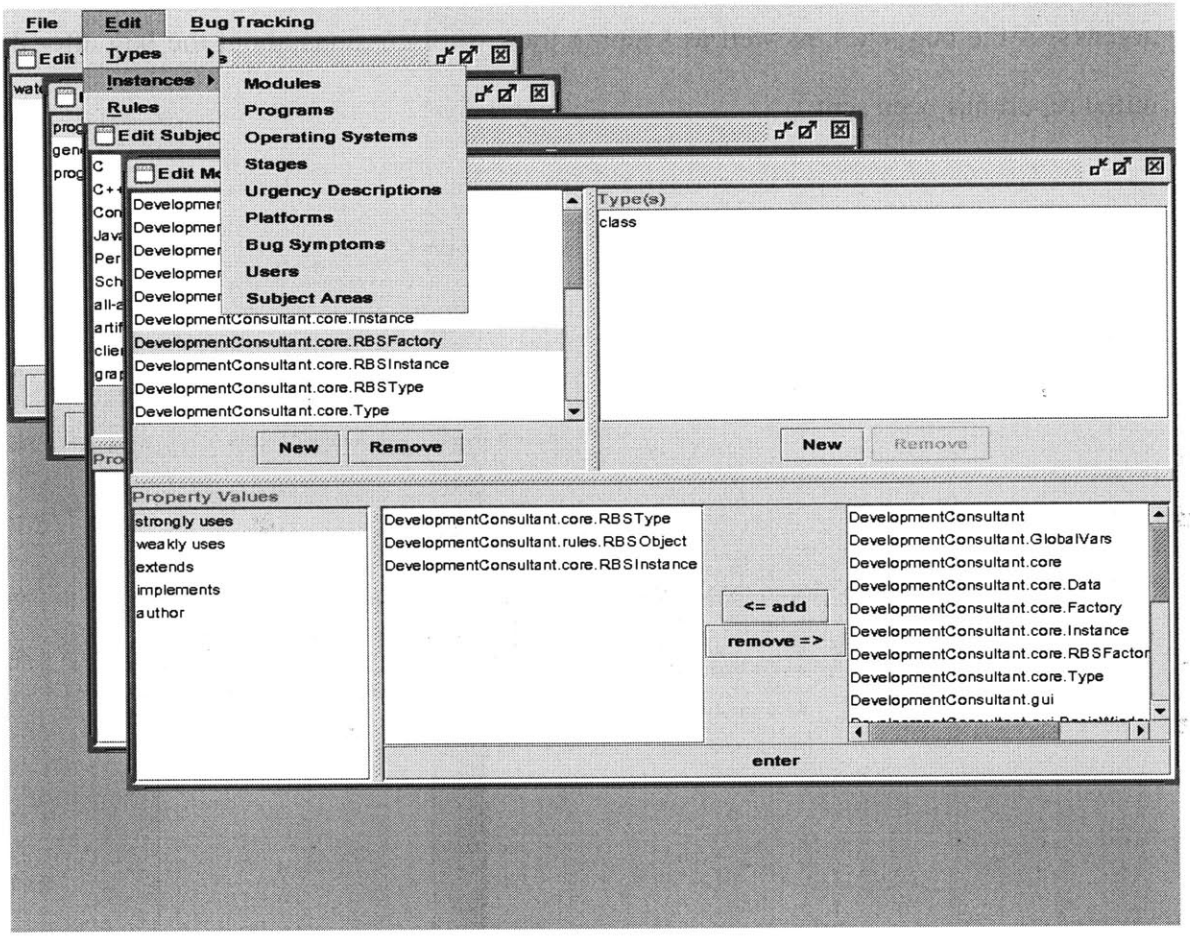


Figure 3.5: Screenshot of entire application

3.4 Bug tracking application

So far we have discussed the specifics of the framework for storing information in our system. Now we can expand upon this framework of information storage to provide useful applications. For my implementation, I chose to create an application for the intelligent reporting and tracking of bugs - a useful application, since bugs are always to be found in the software industry.

This prototype implementation consists of two parts: a mechanism for reporting bugs and one for viewing previously submitted bug reports. A real-world implementation would likely have more features added to the bug viewing half, letting the user search the bug reports by content or particular criteria (such as which module was affected or the urgency of the bug, etc.), as well as keeping track of discussions about the bug after the initial report has been sent.

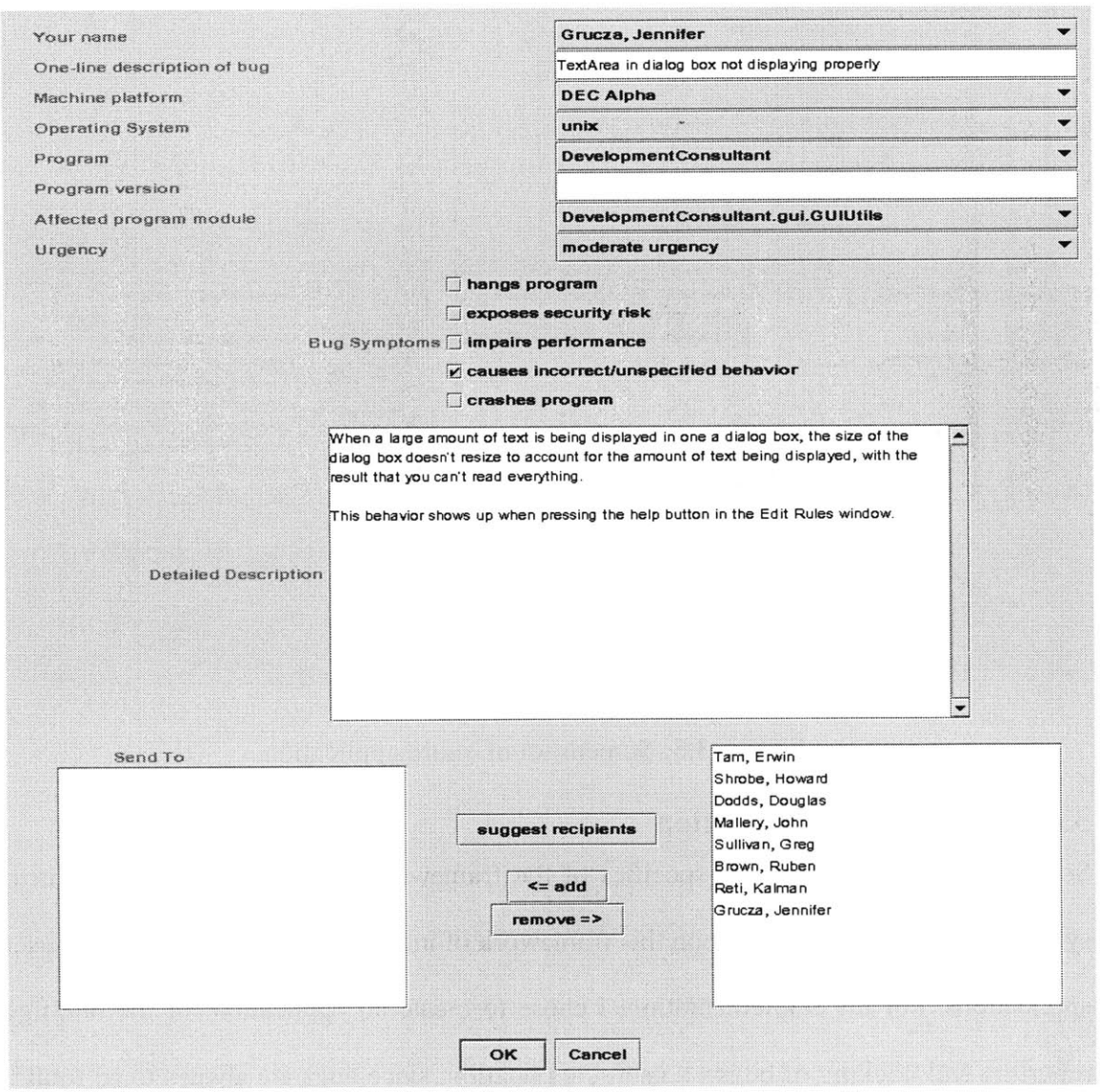


Figure 3.6: Reporting a bug

Figure 3.6 shows the window displayed to the user when they choose the "Report a Bug" option from the "Bug Tracking" menu (see figure 3.5 for a view of the main menu bar). The user is asked to provide their name, chosen from a pull down menu of all users in the system, along with specific information about the bug they are reporting: a brief description of the bug, the machine platform and operating system in use when the bug occurred, the program in which the bug occurred, the version number of the program, the affected module, a description of how urgent the bug situation is, an enumeration of the symptoms caused by the bug, as well as a detailed description of the bug and what was happening when it occurred.

The bottom part of the window is where the user indicates who the bug report should be sent to. There are two ways of doing this. The first way is for the user to manually choose from the list on the right in the case where they know exactly where they want the report sent. The second way is by taking advantage of the rule based system. This is where the real advantage of this system lies. A user may not know which people are best to send a bug report to, and even if they have a good idea of who would best resolve the problem, they may inadvertently miss others who would contribute greatly to such a discussion, and likely help resolve it in a more timely manner. By pressing the "suggest recipients" button, the user can get the system to automatically determine the best possible choices to send the bug report to, depending on what values were entered by the user, as well as what knowledge resides inside the system.

An interesting feature of the demo system's rule collection is that, depending on how urgent the user indicates the bug report is, the rule-based system will choose varying numbers of recipients. If the bug is very urgent, the system will try to find more people to help, whereas if the urgency is very low, the system will not look for as many recipients. This way fewer people will be bothered with relatively inconsequential bugs, but if a

major problem is exposed, a maximum of manpower (of those who would be knowledgeable about the particular problem) is applied for a speedier, better solution. However, it is possible for fewer or more recipients to be chosen, despite the recommended number the system strives for. There will be fewer if the system just cannot find enough people with applicable skills. There can be more because of the way the rules are set up. Certain characteristics indicate that a user should be a "definite participant" in the discussion. For example, if the user authored a module, and a bug report was submitted with an error in that module, we would definitely want the author of that module to be involved in resolving the problem. Other rules simply find "possible participants" and it is from this pool of possible participants that we try to fill out our number of recipients to the desired amount. The members in this pool are ranked by giving each a score which is increased with each match between the person's abilities and those required to resolve the bug. These scores are weighted in importance. However, if we already have our desired number of recipients from the "definite participant" category, we don't necessarily need to include those in the "possible" category.

3.5 Demo knowledge base

The knowledge base chosen to populate the system for the purposes of demonstration was derived from the environment in which the author was working. Thus, the "Programs" factory contained the Instance "DevelopmentConsultant" while the "Modules" factory contained all the modules shown in figure 3.1 (as well as a couple of utility classes which were left out of the module dependency diagram). Likewise, the "Users" factory was populated with the fellow members of the Knowledge-Based Collaboration Webs group at the MIT AI Lab, and so on.

Knowledge about software development environments in general, including the framework of Factories, Types, and their properties, as well as the rules used for the bug

tracking application, was gathered from experience as well as discussions with various software developers. There is undoubtedly much knowledge about software development which has not been included in this demo knowledge-base, but adding such knowledge is not a difficult task (merely a time-intensive one).

Please refer to section 7 for a listing of the knowledge contained in this demo system.

4 Example

To better illustrate the mechanism used by the system to determine the best recipients for a bug report, we will present an example. Let us take the example of a user who has run across a bug in the user interface of a previous version of DevelopmentConsultant. He has discovered that when dialog boxes are displayed with large amounts of text, the window doesn't react by increasing its size, and the result is that not all the text is visible. He selects the "Report a Bug" menu item, and fills out the bug report dialog with the following pertinent information (along with a more detailed written description for human readers):

- machine platform = UNIX
- operating system = unix
- affected program = DevelopmentConsultant
- affected module = DevelopmentConsultant.gui.GUIUtils
- urgency = "minor problem, not urgent"
- symptoms = "causes incorrect/unspecified behavior"

He's not sure who this bug report should be sent to, so he presses the "suggest recipients" button. The system responds by adding three names to the list. But how did it determine that these people would be able to help resolve the bug? Let us delve into the rule-based system, and follow its workings as rules are fired and facts asserted.

The engine starts by giving precedence to rules with the highest salience. In this case, we initialize each user with a score, which will be used to rate them with later.

```
FIRE 1 assign-user-score f-364, f-354
 ==> f-456 (score "Reti, Kalman" 0)
FIRE 2 assign-user-score f-350, f-335
 ==> f-457 (score "Brown, Ruben" 0)
FIRE 3 assign-user-score f-331, f-325
 ==> f-458 (score "Sullivan, Greg" 0)
FIRE 4 assign-user-score f-320, f-309
 ==> f-459 (score "Mallery, John" 0)
FIRE 5 assign-user-score f-299, f-251
```

```
 ==> f-460 (score "Grucza, Jennifer" 0)
FIRE 6 assign-user-score f-247, f-241
 ==> f-461 (score "Dodds, Douglas" 0)
FIRE 7 assign-user-score f-235, f-215
 ==> f-462 (score "Shrobe, Howard" 0)
FIRE 8 assign-user-score f-211, f-201
 ==> f-463 (score "Tam, Erwin" 0)
```

Now the engine begins to look for more rules which are ready to fire. The "exceptionally-skilled" rule looks for matches between the subject areas associated with the program affected by the bug and the subject areas in which users are exceptionally skilled. For example, the line below beginning with "FIRE 9" shown below has matched the subject area "object-oriented programming" with both the program "DevelopmentConsultant" and the user "Reti, Kalman."

```
FIRE 9 exceptionally-skilled f-354,, f-436, f-367, f-452, f-433, f-456
 ==> f-464 (exceptionally-skilled-triggered "Reti, Kalman")
 ==> f-465 (possible-participant "Reti, Kalman")
 ==> f-466 (score "Reti, Kalman" 4)
FIRE 10 exceptionally-skilled f-309,, f-436, f-323, f-452, f-433, f-459
 ==> f-467 (exceptionally-skilled-triggered "Mallery, John")
 ==> f-468 (possible-participant "Mallery, John")
 ==> f-469 (score "Mallery, John" 4)
```

The moderately-skilled-at and very-skilled rules are similar to the exceptionally-skilled rule. The "FIRE 11" shows that we have found a match between the subject area values of "Grucza, Jennifer"'s "moderately skilled at" property, and "Development Consultant"'s "involves subject area" property. In this example, the match is that of "graphical user interface."

As you can see from the these preceding examples, each of these rules, when finding a match, increases the score of the user involved by an appropriate amount. If the user is exceptionally skilled, they get four points added to their score, and if they have only basic skills, one point is added.

```
FIRE 11 moderately-skilled-at f-251,, f-437, f-300, f-452, f-433, f-460
 ==> f-470 (moderately-skilled-at-triggered "Grucza, Jennifer")
 ==> f-471 (possible-participant)
 ==> f-472 (score "Grucza, Jennifer" 2)
```

```
FIRE 12 very-skilled f-251,, f-438, f-291, f-452, f-433, f-472
 ==> f-473 (very-skilled-triggered "Grucza, Jennifer")
 ==> f-474 (possible-participant "Grucza, Jennifer")
 ==> f-475 (score "Grucza, Jennifer" 5)
FIRE 13 exceptionally-skilled f-215,, f-438, f-238, f-452, f-433, f-462
 ==> f-476 (exceptionally-skilled-triggered "Shrobe, Howard")
 ==> f-477 (possible-participant "Shrobe, Howard")
 ==> f-478 (score "Shrobe, Howard" 4)
```

The next two rule firings are those having to do with managers. The first is fired because Howard Shrobe is the manager of Jennifer Grucza, who *worked on* the affected module. The second fires because Howard Shrobe is the manager of Jennifer Grucza, who *authored* the affected module. In this case, the users involved are the same, but in a larger system there would likely be more and different matches.

```
FIRE 14 manager-responsibility-module f-215,, f-287, f-221, f-251, f-453, f-478
 ==> f-479 (manager-responsibility-module-triggered "Shrobe, Howard")
 ==> f-480 (score "Shrobe, Howard" 7)
FIRE 15 manager-module-author f-215,, f-453, f-187, f-189, f-221, f-480
 ==> f-481 (manager-module-author-triggered "Shrobe, Howard")
 ==> f-482 (score "Shrobe, Howard" 10)
```

Next a rule has fired because it has found somebody who should be a definite participant in the bug-discussion. Having worked on the particular module in question means this user would be of invaluable help.

```
FIRE 16 worked-on-module f-287, f-453, f-251
 ==> f-483 (definite-participant "Grucza, Jennifer")
FIRE 17 worked-on-program f-261, f-452, f-251
```

The following collection of rule firings serve the purpose of ranking all the possible participants we have found. Users are ranked by the scores which have been cumulatively added up as each of the rules described above have fired.

```
FIRE 18 find-possible-participants f-477, f-482
FIRE 19 already-participant f-474, f-483
 <== f-474 (possible-participant "Grucza, Jennifer")
FIRE 20 find-possible-participants f-477, f-480
FIRE 21 find-possible-participants f-477, f-478
FIRE 22 find-possible-participants f-477, f-462
FIRE 23 find-possible-participants f-468, f-469
FIRE 24 find-possible-participants f-465, f-466
FIRE 25 find-possible-participants f-468, f-459
FIRE 26 find-possible-participants f-465, f-456
```

```
FIRE 27 module-author f-189, f-453, f-187
FIRE 28 find-participants f-483
```

Since the bug has been specified as not being urgent, the rule "not-urgent" fires, which tells us that we'd like 3 participants in this discussion - enough for some opposing ideas, but not enough to waste resources. The rule "fix-number-participants" then looks to see if we have enough definite-participants. If so, it would stop, satisfied. If not, as in this case where there is only one definite-participant, it would take the highest-ranked possible-participants until the desired number of participants have been met. In this case, the system found that "Reti, Kalman" and "Mallery, John" should round out the three person discussion.

```
FIRE 29 not-urgent f-454
 ==> f-484 (need-number-participants 3)
FIRE 30 fix-number-participants f-484,
```

Once it has come to a decision as to which users to send the bug report to, the user interface updates the recipient list by adding these selected users.

5 Conclusion and Future Directions

5.1 Evaluation

The implementation written has proven to fulfil the goal of being easily extensible. Evaluation of the system using information about the members of the KBCW group at the AI Lab as well as the modules of code making up the implementation itself have shown that the rule-based system and the bug reporting application behave as specified, and are able to intelligently choose participants for discussions about resolving bugs, based on the information submitted in the bug report by the user. The next step would be to test the system on a broader knowledge base, with a large number of users and a large code base (such as the CL-HTTP web server at over 50,000 lines of code), over a longer period of time, to determine the performance of the system in more stressful conditions, as well as the effectiveness of the bug reporting application with the current set of rules.

The following sections describe ways in which the program itself could be expanded and improved in various ways.

5.2 Improve user interface

While the current user interface is functional and reasonably understandable, it has room for improvement. A small future project might be to construct various alternative mock user interfaces, and conduct a survey to determine which elements make the interface most easy to understand and use. The current GUI is somewhat biased in favor of predominantly mouse-using users rather than those who prefer to use only keyboard. A good interface design should make it easy for either approach.

5.3 New file format

DevelopmentConsultant uses Java object serialization for the storage of system state. Unfortunately, this method has the drawback that modification of the program's core

classes renders the new version incapable of reading the previous version's saved state. To prevent this from happening, a consistent file format should be developed - one which will remain the same from one version to another. In other words, it should be built upon the specifications of the program, so that even if the implementation changes, the file format will stay the same.

5.4 Expand knowledge base

5.4.1 *Expanded framework*

The current state of the knowledge base contained in the system provides much information about software development environments, but there is more information which could be added to the system to make it smarter and more complete. For example, currently there is only one type of software program known to the system - a "default" type. We might wish to further categorize, and upon doing so, include properties specific to each type of program (perhaps we would want to keep track of the hardware demands of all programs, or qualify what level of graphics a computer game provided, for instance). Another example of an expansion we might want to make is to add new Types to the "Stages" Factory. Many companies don't use the traditional waterfall model to describe their planning, and it could be useful to add knowledge of these other models to the system.

Luckily, adding such knowledge is extremely simple since it can all be done by the user through DevelopmentConsultant's graphical interface.

5.4.2 *Additional rules*

Almost as easy as adding to the framework described in the previous section is adding new rules to DevelopmentConsultant's rule-based system. Unfortunately, codifying the extent of human knowledge about how software development environments work is a somewhat long and tedious task, not to mention the difficulty in knowing the best way to

represent such knowledge in rule form and how to decide which pieces of knowledge are relevant or important, or even accurate. The rules which have been provided in the demonstration system form only a small subset of a desired, comprehensive set of rules pertaining to software development. Namely, those which have been included are those which are most convenient and informative for intelligent bug reporting, the selected focus of this project.

5.5 Improve bug tracking

As just stated, the main focus of the demonstration implementation was bug reporting, but intelligent bug reporting in and of itself is only part of the problem. In addition, it is necessary to be able to store the bug reports, view them at a later time, as well as be able to track discussion leading to their resolution as well as their current status. This system covers the first two. A next step would be to either include a user interface for discussion (relegating the current bug reporting mechanism to a notification service, after which all discussion would take place through the application), or to incorporate the email discussion directly into the application. This might be the easiest solution, and perhaps most convenient for users who are used to email as the communication medium of choice. An email address could be created solely for receiving emails of such discussions, and the application could be set up to receive these emails and automatically archive them. Additionally, the GUI could be expanded to include a flag indicating each bug's current status (and a way for that flag to be set by authorized users).

5.6 Convert to a distributed system

The prototype implementation described in this thesis is a non-distributed program. Strictly speaking, it is possible for more than one user to use the program at once, but the program was not designed with this in mind, and it is inevitable that information would be

lost if it were to be used in this manner. For example, take two users, Bob and Mary, who are running DevelopmentConsultant at the same time. Bob makes some modifications to various Types and Instances and selects "Save" from the "File" menu. Mary, meanwhile, has submitted a bug report and saved state herself. If Mary saved after Bob, all Bob's modifications are lost, and if Bob saved later than Mary, the bug report she sent is not archived (although the email is still sent out, of course). So currently the system is only designed to support one person's use at a time. Obviously, this is not desirable in a software company made up of hundreds or thousands of people.

Therefore, a very beneficial change to this program would be to convert it to a distributed system. A central server would store all of the data, and field requests from individual clients to modify that data, report bugs, and so on. Such a system could also be designed to limit the accessibility of certain functions to authorized personnel (see sections 3.1 and 3.3), which would be a desirable feature for many organizations.

Converting the existing system over to a client/server model should not be too difficult a task, as there already exists the necessary division between the core classes, which hold the data, and the GUI classes, which make that data available to the user. The core classes would reside in the server half, with the GUI in the client. A middle layer would be added to handle the networking, and some additional modification would be necessary to account for lost packets and various network errors, but such a modification would be very feasible, especially considering there are existing Java tools for helping create such a system.

5.7 Add new applications

Section 2.2 outlined some possible applications which would be useful in such a large-scale administrative tool. Especially interesting to incorporate would be Christopher

Vincent's work in code unit annotation, as well as an application for running and keeping track of regression tests.

5.8 Create new domains

As was mentioned in section 3.1, the domain chosen for this project was software development environments, but effort was made to construct the design in order to make the framework largely general. Interesting projects would include adapting the system to other domains, such as the military, consumer marketing and purchasing, or just other commercial industries in addition to the software industry.

5.9 Incorporate other existing administrative applications

Section 2 described an ideal environment to work in, where all the various administrative programs used in an organization would be tied together with a system such as this one, so that site-specific information wouldn't have to be entered manually, and updated manually as well. Developing such a web of applications, all interconnected, is a long-term goal, with many complexities involved in implementing it, but it is a goal to be strived for.

6 References

[1] Arnold, Ken and James Gosling. *The Java Programming Language, Second Edition.* Addison-Wesley, 1998.

[2] Cooper, J. "Software Development Management Planning," *IEEE Transactions on Software Engineering*, vol. se-10, no. 1, Jan. 1984, p. 22-26.

[3] Davis, R., H. Shrobe, P. Szolovits. "What is a Knowledge Representation?" *AI Magazine*, 14(1):17-33, 1993.

[4] Dolina, George. *Intelligent Resource Allocation in Distributed Collaboration.* MIT, May 1999

[5] Eckstein, Robert, Marc Loy, and Dave Wood. *Java Swing.* O'Reilly, Dec. 1998.

[6] Friedman-Hill, Ernest. *Jess, the Java Expert System Shell.* Sandia National Laboratories, July 1999. **http://herzberg.ca.sandia.gov/jess/main.html**

[7] Giarratano, Joseph C. *CLIPS User's Guide*, version 6.10. Software Technology Branch NASA/Lyndon B. Johnson Space Center, Aug. 1998.

[8] Jackson, Daniel. Module Dependency Diagram and Object Models Crib Sheets. Feb. 1999. **http://web.mit.edu/6.170/www/handouts/07-mdd-crib.ps & http://web.mit.edu/6.170/www/handouts/08-om-crib.ps**

[9] Kim, W., Y. Suh, A.B. Whinston. "An IBIS and object-oriented approach to scientific research data management," *Journal of Systems & Software*, vol. 23, no. 2. Nov. 1993, p. 183-97.

[10] "Program Analysis and Metrics." Software Methods and Tools, 1997-9. **http://www.methods-tools.com/tools/other.html**

[11] *Rule of Eight EasyMail.* Rule of Eight, 1999. **http://www.rule-of-eight.com/components/easymail/**

[12] Scacchi, W. "Managing Software Engineering Projects: A Social Analysis," *IEEE Transactions on Software Engineering*, vol. se-10, no. 1, Jan.1984, p. 49-59.

[13] Vincent, Christopher. *DATE: A Framework for Supporting Design Artifact Tracking and Evolution.* MIT, May 1999.

7 Demo Knowledge Base

```
Contents of DevelopmentConsultant's Knowledge Base
==================================================
```

RULES

```
(defrule find-possible-participants
  (possible-participant ?user)
  (score ?user ?score)
  =>
  (bind ?vector (fetch "possible-participants"))
  (bind ?hashtable (fetch "score-table"))
  (call ?hashtable put ?user (new java.lang.Integer ?score))
  (bind ?i 0)
  (while (< ?i (call ?vector size))
       (bind ?x (call ?vector elementAt ?i))
       (if (> ?score
              (call ?hashtable get ?x))
        then (call ?vector insertElementAt ?user ?i)
             (bind ?i (+ (call ?vector size) 10))
        else (bind ?i (+ 1 ?i))))
  (if (= ?i 0)
   then (call ?vector addElement ?user)))


(defrule fix-number-participants
  (declare (salience -100))
  (need-number-participants ?number)
  =>
  (bind ?needed (- ?number
                   (call (fetch "participants") size)))
  (bind ?vector (fetch "possible-participants"))
  (bind ?result (fetch "participants"))
  (bind ?i 0)
  (while (and (< ?i (call ?vector size))
              (> ?needed 0))
       (call ?result addElement
              (call ?vector elementAt ?i))
       (bind ?i (+ ?i 1))
       (bind ?needed (- ?needed 1))))

(defrule severly-urgent
  (bug-urgency "severely urgent, progress impossible")
  =>
  (assert (need-number-participants 15)))

(defrule worked-on-module
  (instance-has-property-value
       (instance ?user)
       (property "worked on module")
       (value ?module))
  (bug-module ?module)
  (instance-identifier
       (identifier ?user-string)
       (value ?user))
  =>
  (assert (definite-participant ?user-string)))
```

```
(defrule manager-propagation
  (instance-has-property-value
        (instance ?high-manager)
        (property "manages")
        (value ?middle-manager))
  (instance-has-property-value
        (instance ?middle-manager)
        (property "manages")
        (value ?user))
  =>
  (assert (instance-has-property-value
              (instance ?high-manager)
              (property "manages")
              (value ?user))))

(defrule manager-responsibility-module
  (instance-identifier
        (identifier ?manager-string)
        (value ?manager))
  (not (manager-responsibility-module-triggered ?manager-string))
  (instance-has-property-value
        (instance ?user)
        (property "worked on module")
        (value ?module))
  (instance-has-property-value
        (instance ?manager)
        (property "manages")
        (value ?user-string))
  (instance-identifier
        (identifier ?user-string)
        (value ?user))
  (bug-module ?module)
  (score ?manager-string ?score)
  =>
  (assert (manager-responsibility-module-triggered ?manager-string))
  (bind ?s (+ ?score 3))
  (assert (score ?manager-string ?s))
  (assert (possible-participant ?manager-string)))

(defrule exceptionally-skilled
  (instance-identifier
        (identifier ?user-string)
        (value ?user))
  (not (exceptionally-skilled-triggered ?user-string))
  (instance-has-property-value
        (instance ?program)
        (property "involves subject area")
        (value ?area-string))
  (instance-has-property-value
        (instance ?user)
        (property "exceptionally skilled at")
        (value ?area-string))
  (bug-program ?program-string)
  (instance-identifier
        (identifier ?program-string)
        (value ?program))
  (score ?user-string ?score)
  =>
```

```
  (assert (exceptionally-skilled-triggered ?user-string))
  (bind ?s (+ ?score 4))
  (assert (possible-participant ?user-string))
  (assert (score ?user-string ?s)))

(defrule basic-skills
  (instance-identifier
       (identifier ?user-string)
       (value ?user))
  (not (basic-skills-triggered ?user-string))
  (instance-has-property-value
       (instance ?program)
       (property "involves subject area")
       (value ?area-string))
  (instance-has-property-value
       (instance ?user)
       (property "has basic skills in")
       (value ?area-string))
  (bug-program ?program-string)
  (instance-identifier
       (identifier ?program-string)
       (value ?program))
  (score ?user-string ?score)
  =>
  (assert (basic-skills-triggered ?user-string))
  (bind ?s (+ ?score 1))
  (assert (possible-participant ?user-string))
  (assert (score ?user-string ?s)))

(store "possible-participants" (new java.util.Vector))
(store "score-table" (new java.util.Hashtable))

(defrule already-participant
  ?x <- (possible-participant ?user)
  (definite-participant ?user)
  =>
  (retract ?x))

(defrule manager-module-author
  (instance-identifier
       (identifier ?manager-string)
       (value ?manager))
  (not (manager-module-author-triggered ?manager-string))
  (bug-module ?module-string)
  (instance-identifier
       (identifier ?module-string)
       (value ?module))
  (instance-has-property-value
       (instance ?module)
       (property "author")
       (value ?author))
  (instance-has-property-value
       (instance ?manager)
       (property "manages")
       (value ?author))
  (score ?manager-string ?score)
  =>
  (assert (manager-module-author-triggered ?manager-string))
  (bind ?s (+ ?score 3))
```

```
  (assert (score ?manager-string ?s))
  (assert (possible-participant ?manager-string)))

(defrule assign-user-score
  (declare (salience 100))
  (instance-has-property-value
        (instance ?user)
        (property "email address"))
  (instance-identifier
        (identifier ?user-string)
        (value ?user))
  =>
  (assert (score ?user-string 0)))

(defrule very-skilled
  (instance-identifier
        (identifier ?user-string)
        (value ?user))
  (not (very-skilled-triggered ?user-string))
  (instance-has-property-value
        (instance ?program)
        (property "involves subject area")
        (value ?area-string))
  (instance-has-property-value
        (instance ?user)
        (property "very skilled at")
        (value ?area-string))
  (bug-program ?program-string)
  (instance-identifier
        (identifier ?program-string)
        (value ?program))
  (score ?user-string ?score)
  =>
  (assert (very-skilled-triggered ?user-string))
  (bind ?s (+ ?score 3))
  (assert (possible-participant ?user-string))
  (assert (score ?user-string ?s)))

(defrule moderately-skilled-at
  (instance-identifier
        (identifier ?user-string)
        (value ?user))
  (not (moderately-skilled-at-triggered ?user-string))
  (instance-has-property-value
        (instance ?program)
        (property "involves subject area")
        (value ?area-string))
  (instance-has-property-value
        (instance ?user)
        (property "moderately skilled at")
        (value ?area-string))
  (bug-program ?program-string)
  (instance-identifier
        (identifier ?program-string)
        (value ?program))
  (score ?user-string ?score)
  =>
  (assert (moderately-skilled-at-triggered ?user-string))
  (bind ?s (+ ?score 2))
```

```
  (assert (possible-participant))
  (assert (score ?user-string ?s)))

(defrule moderate-urgency
  (bug-urgency "moderate urgency")
  =>
  (assert (need-number-participants 5)))

(defrule not-urgent
  (bug-urgency "minor problem, not urgent")
  =>
  (assert (need-number-participants 3)))

(defrule module-author
  (instance-has-property-value
       (instance ?module)
       (property "author")
       (value ?author-string))
  (bug-module ?module-string)
  (instance-identifier
       (identifier ?module-string)
       (value ?module))
  =>
  (assert (definite-participant ?author-string)))

(defrule worked-on-program
  (instance-has-property-value
       (instance ?user)
       (property "worked on program")
       (value ?program))
  (bug-program ?program)
  (instance-identifier
       (identifier ?user-string)
       (value ?user))
  =>
  (assert (possible-participant ?user-string)))
```

FACTS

Subject Areas:

Types: programming concept. general skill. programming language.
Instances:

```
       slow but steady wins the race
              types: general skill.
              properties:

       client/server
              types: programming concept.
              properties:

       Common Lisp
              types: programming language.
              properties:

       Java
              types: programming language.
              properties:
```

```
        graphical user interface
                types: programming concept.
                properties:

        object-oriented programming
                types: programming concept.
                properties:

        Perl
                types: programming language.
                properties:

        Scheme
                types: programming language.
                properties:

        C
                types: programming language.
                properties:

        all-around guru
                types: general skill.
                properties:

        quick learner
                types: general skill.
                properties:

        artificial intelligence
                types: programming concept.
                properties:

        C++
                types: programming language.
                properties:

Bug Symptoms:

Types: default.
Instances:
        hangs program
                types: default.
                properties:

        exposes security risk
                types: default.
                properties:

        impairs performance
                types: default.
                properties:

        causes incorrect/unspecified behavior
                types: default.
                properties:

        crashes program
                types: default.
```

```
                properties:

Platforms:

Types: default.
Instances:
        DEC Alpha
                types: default.
                properties:

        Sun
                types: default.
                properties:

        Macintosh
                types: default.
                properties:

        Symbolics Lisp Machine
                types: default.
                properties:

        PC
                types: default.
                properties:

        UNIX
                types: default.
                properties:

Modules:

Types: interface. package. class.
Instances:
        DevelopmentConsultant.core
                types: package.
                properties:
                        author Grucza, Jennifer.
                        contains moduleDevelopmentConsultant.core.Instance.
DevelopmentConsultant.core.Type. DevelopmentConsultant.core.RBSType.
DevelopmentConsultant.core.Factory. DevelopmentConsultant.core.RBSInstance.
DevelopmentConsultant.core.Data. DevelopmentConsultant.core.RBSFactory.

        DevelopmentConsultant.rules
                types: package.
                properties:
                        related toartificial intelligence.
                        author Grucza, Jennifer.
                        contains moduleDevelopmentConsultant.rules.RBSObject.
DevelopmentConsultant.rules.RBSEngine.

        DevelopmentConsultant.GlobalVars
                types: class.
                properties:
                        author Grucza, Jennifer.

        DevelopmentConsultant.core.Factory
                types: class.
                properties:
```

```
                        weakly usesDevelopmentConsultant.core.Instance.
DevelopmentConsultant.core.Type.
                        author Grucza, Jennifer.

        DevelopmentConsultant.core.RBSType
                types: class.
                properties:
                        extends DevelopmentConsultant.rules.RBSObject.
DevelopmentConsultant.core.Type.
                        author Grucza, Jennifer.

        DevelopmentConsultant.rules.RBSObject
                types: interface.
                properties:
                        author Grucza, Jennifer.

        DevelopmentConsultant.core.Type
                types: class.
                properties:
                        author Grucza, Jennifer.

        DevelopmentConsultant.core.Instance
                types: class.
                properties:
                        author Grucza, Jennifer.

        DevelopmentConsultant.gui.EditInstancesInternalFrame
                types: class.
                properties:
                        strongly usesDevelopmentConsultant.core.Factory.
DevelopmentConsultant.core.Data. DevelopmentConsultant.core.Instance.
DevelopmentConsultant.gui.Utilities. DevelopmentConsultant.gui.GUIUtils.
                        author Grucza, Jennifer.

        DevelopmentConsultant.gui.Utilities
                types: class.
                properties:
                        author Grucza, Jennifer.

        DevelopmentConsultant.gui.ProgramDesktop
                types: class.
                properties:
                        strongly usesDevelopmentConsultant.gui.ReportBugAction.
DevelopmentConsultant.core.Data.
DevelopmentConsultant.gui.EditInstancesInternalFrame.
DevelopmentConsultant.gui.ViewBugsInternalFrame.
DevelopmentConsultant.gui.EditTypesInternalFrame.
DevelopmentConsultant.gui.EditRulesInternalFrame.
DevelopmentConsultant.gui.BasicWindowMonitor.
DevelopmentConsultant.gui.GUIUtils.
                        author Grucza, Jennifer.

        DevelopmentConsultant.gui.ReportBugAction
                types: class.
                properties:
                        strongly usesDevelopmentConsultant.core.Data.
DevelopmentConsultant.core.Factory. DevelopmentConsultant.core.Instance.
DevelopmentConsultant.rules.RBSEngine. DevelopmentConsultant.gui.Utilities.
DevelopmentConsultant.gui.GUIUtils.
```

```
                    author Grucza, Jennifer.

        DevelopmentConsultant
                types: package.
                properties:
                        related toobject-oriented programming.
                        author Grucza, Jennifer.
                        contains moduleDevelopmentConsultant.rules.
DevelopmentConsultant.core. DevelopmentConsultant.gui.
DevelopmentConsultant.GlobalVars.

        DevelopmentConsultant.gui
                types: package.
                properties:
                        related tographical user interface.
                        author Grucza, Jennifer.
                        contains module
DevelopmentConsultant.gui.EditInstancesInternalFrame.
DevelopmentConsultant.gui.EditTypesInternalFrame.
DevelopmentConsultant.gui.Main. DevelopmentConsultant.gui.GUIUtils.
DevelopmentConsultant.gui.EditRulesInternalFrame.
DevelopmentConsultant.gui.BasicWindowMonitor.

        DevelopmentConsultant.core.Data
                types: class.
                properties:
                        strongly usesDevelopmentConsultant.core.RBSFactory.
                        author Grucza, Jennifer.

        DevelopmentConsultant.core.RBSInstance
                types: class.
                properties:
                        extends DevelopmentConsultant.rules.RBSObject.
DevelopmentConsultant.core.Instance.
                        strongly uses
                        author Grucza, Jennifer.

        DevelopmentConsultant.rules.RBSEngine
                types: class.
                properties:
                        strongly usesDevelopmentConsultant.core.Data.
DevelopmentConsultant.core.RBSFactory.
                        author Grucza, Jennifer.

        DevelopmentConsultant.gui.EditTypesInternalFrame
                types: class.
                properties:
                        strongly usesDevelopmentConsultant.gui.GUIUtils.
DevelopmentConsultant.core.Factory. DevelopmentConsultant.core.Data.
DevelopmentConsultant.core.Type.
                        author Grucza, Jennifer.

        DevelopmentConsultant.gui.ViewBugsInternalFrame
                types: class.
                properties:
                        strongly usesDevelopmentConsultant.core.Data.
DevelopmentConsultant.gui.Utilities.
                        author Grucza, Jennifer.
```

```
        DevelopmentConsultant.core.RBSFactory
                types: class.
                properties:
                        extends DevelopmentConsultant.rules.RBSObject.
                        author Grucza, Jennifer.
                        weakly usesDevelopmentConsultant.core.Type.
DevelopmentConsultant.core.Instance.
                        strongly usesDevelopmentConsultant.core.RBSType.
DevelopmentConsultant.rules.RBSObject. DevelopmentConsultant.core.RBSInstance.

        DevelopmentConsultant.gui.BasicWindowMonitor
                types: class.
                properties:
                        author Grucza, Jennifer.

        DevelopmentConsultant.gui.EditRulesInternalFrame
                types: class.
                properties:
                        strongly usesDevelopmentConsultant.gui.Utilities.
DevelopmentConsultant.gui.GUIUtils. DevelopmentConsultant.core.Data.
                        author Grucza, Jennifer.

        DevelopmentConsultant.gui.GUIUtils
                types: class.
                properties:
                        author Grucza, Jennifer.

        DevelopmentConsultant.gui.Main
                types: class.
                properties:
                        strongly usesDevelopmentConsultant.gui.ProgramDesktop.
                        author Grucza, Jennifer.

Users:

Types: quality assurance engineer. documentation writer. programmer. basic user.
customer. developer. project manager.
Instances:
        *omitted except for*
        Grucza, Jennifer
                types: basic user. developer. programmer.
                properties:
                        operating system(s)unix. Windows95. MacOS.
                        worked on programDevelopmentConsultant.
                        home address***
                        middle initialE
                        organizationMIT AI Lab
                        worked on moduleDevelopmentConsultant.core.
DevelopmentConsultant.rules. DevelopmentConsultant.GlobalVars.
DevelopmentConsultant.core.Factory. DevelopmentConsultant.core.RBSType.
DevelopmentConsultant.rules.RBSObject. DevelopmentConsultant.core.Type.
DevelopmentConsultant.core.Instance.
DevelopmentConsultant.gui.EditInstancesInternalFrame.
DevelopmentConsultant.gui.Utilities. DevelopmentConsultant.gui.ProgramDesktop.
DevelopmentConsultant.gui.ReportBugAction.
DevelopmentConsultant. DevelopmentConsultant.gui.
DevelopmentConsultant.core.Data. DevelopmentConsultant.core.RBSInstance.
DevelopmentConsultant.rules.RBSEngine.
DevelopmentConsultant.gui.EditTypesInternalFrame. DevelopmentConsultant.
```

gui.ViewBugsInternalFrame. DevelopmentConsultant.core.RBSFactory.
DevelopmentConsultant.gui.BasicWindowMonitor.
DevelopmentConsultant.gui.EditRulesInternalFrame.
DevelopmentConsultant.gui.GUIUtils. DevelopmentConsultant.gui.Main.
very skilled atJava. object-oriented programming.
artificial intelligence. slow but steady wins the race.
work phone617.253.5043
home phone***
has basic skills inclient/server. C. Common Lisp.
work addressNE43-832
email addressjgrucza@alum.mit.edu
moderately skilled atgraphical user interface. Perl.
Scheme. quick learner.
platform(s)Macintosh. PC. UNIX.
last nameGrucza
first nameJennifer

Stages:

Types: waterfall model.
Instances:
maintenance
types: waterfall model.
properties:
after testing. design. deployment. requirements analysis.
brainstorming. documentation. implementation.

testing
types: waterfall model.
properties:
after design. requirements analysis. brainstorming.
implementation.

design
types: waterfall model.
properties:
after requirements analysis. brainstorming.

deployment
types: waterfall model.
properties:
after testing. design. requirements analysis.
brainstorming. documentation. implementation.

requirements analysis
types: waterfall model.
properties:
after brainstorming.

brainstorming
types: waterfall model.
properties:

documentation
types: waterfall model.
properties:
after testing. design. requirements analysis.
brainstorming. implementation.

```
        implementation
                types: waterfall model.
                properties:
                        after   design. requirements analysis. brainstorming.

Operating Systems:

Types: default.
Instances:
        unix
                types: default.
                properties:

        WindowsNT
                types: default.
                properties:

        genera
                types: default.
                properties:

        Windows95
                types: default.
                properties:

        Windows98
                types: default.
                properties:

        MacOS
                types: default.
                properties:

        Linux
                types: default.
                properties:

        Solaris
                types: default.
                properties:

Programs:

Types: default.
Instances:
        DevelopmentConsultant
                types: default.
                properties:
                        descriptionAn extensible, intelligent system for applying
software development knowledge
                        involves subject areaobject-oriented programming.
graphical user interface. artificial intelligence.
                        programming language usedJava.
                        contact Grucza, Jennifer.

Urgency Descriptions:

Types: bugs.
Instances:
```

```
severely urgent, progress impossible
        types: bugs.
        properties:

minor problem, not urgent
        types: bugs.
        properties:

moderate urgency
        types: bugs.
        properties:
```

BUG REPORTS

The following bug report was created and sent using DevelopmentConsultant

```
Sender:  Grucza, Jennifer
Date of report:  Wed Aug 18 02:57:13 EDT 1999
Urgency:  minor problem, not urgent
Platform:  UNIX
Operating System:  UNIX
Program:  DevelopmentConsultant
Version:
Affected module:  DevelopmentConsultant.gui.ReportBugAction
Bug Symptoms:

Detailed Description:
This is where one would describe the bug in more detail, including what happened
to cause the bug (and what was running at the time, etc.).
```

The following bug report was created and sent using DevelopmentConsultant

```
Sender:  Grucza, Jennifer
Date of report:  Thu Aug 19 11:49:20 EDT 1999
Urgency:  moderate urgency
Platform:  DEC Alpha
Operating System:  DEC Alpha
Program:  DevelopmentConsultant
Version:
Affected module:  DevelopmentConsultant.gui.GUIUtils
Bug Symptoms:
        causes incorrect/unspecified behavior

Detailed Description:
When a large amount of text is being displayed in one a dialog box, the size of
the dialog box doesn't resize to account for the amount of text being displayed,
with the result that you can't read everything.
```

This behavior shows up when pressing the help button in the Edit Rules window.

8 Source Code

8.1 Global variables

8.1.1 *GlobalVars.java*

```
/* Copyright (c) 1999 Jennifer E. Grucza
   All Rights Reserved. */

package DevelopmentConsultant;

public class GlobalVars {
  public static final String saveFilename =
         "/mit/jgrucza/DevelopmentConsultant/DC.state";
  public static final String rulesHelpFilename =
         "/mit/jgrucza/DevelopmentConsultant/rules-help.txt";
  public static final String outgoingMailServer = "mit.mit.edu";
}
```

8.2 Knowledge representation

8.2.1 *core/Data.java*

```
/* Copyright (c) 1999 Jennifer E. Grucza
   All Rights Reserved. */

package DevelopmentConsultant.core;

import java.util.*;

public class Data implements java.io.Serializable {

  /** This class serves as a repository for all the data in the
      system, as well as the collection of rules which are
      used by the Bug Reporting application (and any others
      which might be added in the future. **/

  public Hashtable factories = new Hashtable();
  public Hashtable rules = new Hashtable();
  public Hashtable bugReports = new Hashtable();

  public Data() {
    /** Creates the groups of information "factories" we will
        need.  **/

    factories.put("Users", new RBSFactory());
    factories.put("Subject Areas", new RBSFactory());
    factories.put("Stages", new RBSFactory());
    factories.put("Platforms", new RBSFactory());
    factories.put("Operating Systems", new RBSFactory());
    factories.put("Programs", new RBSFactory());
    factories.put("Modules", new RBSFactory());
    factories.put("Urgency Descriptions", new RBSFactory());
    factories.put("Bug Symptoms", new RBSFactory());
  }

  public String toString() {
```

```
    /** Prints out a representation of the state of the system. **/

    StringBuffer s = new StringBuffer();
    s.append("Contents of DevelopmentConsultant's Knowledge Base\n");
    s.append("==================================================\n\n");

    s.append("RULES\n\n");
    Enumeration enum = rules.keys();
    while (enum.hasMoreElements()) {
      String rule = (String)rules.get(enum.nextElement());
      s.append(rule + "\n\n");
    }

    s.append("\nFACTS\n\n");
    enum = factories.keys();
    while (enum.hasMoreElements()) {
      String f = (String)enum.nextElement();
      s.append(f + ":\n\n");
      s.append(((Factory)factories.get(f)).toString());
    }

    s.append("\nBUG REPORTS\n\n");
    enum = bugReports.keys();
    while (enum.hasMoreElements()) {
      String r = (String)bugReports.get((String)enum.nextElement());
      s.append(r + "\n\n");
    }

    return s.toString();
  }
}
```

8.2.2 *core/Factory.java*

```
/* Copyright (c) 1999 Jennifer E. Grucza
   All Rights Reserved. */

package DevelopmentConsultant.core;

import java.util.*;

public class Factory implements java.io.Serializable {

  /** A Factory represents a group of related information.  It
      stores what different Types are available for this group,
      as well as all of the individual Instances which have
      been created.  **/

  private Hashtable types = new Hashtable();
  private Hashtable instances = new Hashtable();

  public Type getType(String name) throws NoSuchTypeException {
    if (types.containsKey(name)) {
      return (Type)types.get(name);
    }
    else throw new NoSuchTypeException();
  }

  public void addType(String name, Type type) throws TypeExistsException {
```

```
  if (types.containsKey(name)) {
    throw new TypeExistsException();
  }
  else {
    types.put(name, type);
  }
}

public void removeType(String name) {
  types.remove(name);
}

public Vector getTypes() {
  Enumeration e = types.keys();
  Vector v = new Vector();
  while (e.hasMoreElements()) {
    v.addElement((String)e.nextElement());
  }
  return v;
}

public Enumeration types() {
  return types.keys();
}

public Instance getInstance(String name) throws NoSuchInstanceException {
  if (instances.containsKey(name)) {
    return (Instance)instances.get(name);
  }
  else {
    throw new NoSuchInstanceException();
  }
}

public void addInstance(String name, Instance instance)
     throws InstanceExistsException {
  if (instances.containsKey(name)) {
    throw new InstanceExistsException();
  }
  else {
    instances.put(name, instance);
  }
}

public void removeInstance(String name) {
  instances.remove(name);
}

public Vector getInstances() {
  Enumeration e = instances.keys();
  Vector v = new Vector();
  while (e.hasMoreElements()) {
    v.addElement((String)e.nextElement());
  }
  return v;
}

public Enumeration instances() {
  return instances.keys();
```

```
  }

  public String toString() {
    StringBuffer s = new StringBuffer();

    s.append("Types: ");
    Enumeration enum = types();
    while (enum.hasMoreElements()) {
      s.append((String)enum.nextElement() + ". ");
    }
    s.append("\nInstances:\n");
    enum = instances();
    while (enum.hasMoreElements()) {
      String name = (String)enum.nextElement();
      s.append("\t" + name + "\n");
      try {
        s.append(getInstance(name));
      }
      catch (NoSuchInstanceException nsie) {
        s.append("\t\tCould not retrieve Instance data");
      }
    }
    s.append("\n");

    return s.toString();
  }
}
```

8.2.3 *core/RBSFactory.java*

```
/* Copyright (c) 1999 Jennifer E. Grucza
   All Rights Reserved. */

package DevelopmentConsultant.core;

import DevelopmentConsultant.rules.*;
import java.util.*;
import jess.*;

public class RBSFactory extends Factory implements RBSObject {

  private Vector addedTypes = new Vector();
  private Vector addedInstances = new Vector();
  private Hashtable removedTypes = new Hashtable();
  private Hashtable removedInstances = new Hashtable();

  public void createFacts(Rete r) throws JessException {
    try {
      Enumeration enum = super.types();
      while (enum.hasMoreElements()) {
        String t = (String)enum.nextElement();
        RBSType rbst = (RBSType)super.getType(t);
        createTypeID(t, rbst, r);
        rbst.createFacts(r);
      }
      enum = super.instances();
      while (enum.hasMoreElements()) {
        String i = (String)enum.nextElement();
        RBSInstance rbsi = (RBSInstance)super.getInstance(i);
```

```
      createInstanceID(i, rbsi, r);
      rbsi.createFacts(r);
    }
  }
  catch (NoSuchTypeException nste) {
    System.out.println(nste);
  }
  catch (NoSuchInstanceException nsie) {
    System.out.println(nsie);
  }
}

public void updateFacts(Rete r) throws JessException {
  Enumeration enum = super.types();
  while (enum.hasMoreElements()) {
    RBSType t = (RBSType)enum.nextElement();
    t.updateFacts(r);
  }
  enum = super.instances();
  while (enum.hasMoreElements()) {
    RBSInstance inst = (RBSInstance)enum.nextElement();
    inst.updateFacts(r);
  }
  enum = removedTypes.keys();
  while (enum.hasMoreElements()) {
    removeTypeFact((String)enum.nextElement(), r);
  }
  enum = removedInstances.keys();
  while (enum.hasMoreElements()) {
    removeInstanceFact((String)enum.nextElement(), r);
  }

  addedTypes.removeAllElements();
  addedInstances.removeAllElements();
  removedTypes.clear();
  removedInstances.clear();
}

public void deleteFacts(Rete r) throws JessException {
  Enumeration enum = super.types();
  while (enum.hasMoreElements()) {
    removeTypeFact((String)enum.nextElement(), r);
  }
  enum = super.instances();
  while (enum.hasMoreElements()) {
    removeInstanceFact((String)enum.nextElement(), r);
  }
  addedTypes.removeAllElements();
  addedInstances.removeAllElements();
  removedTypes.clear();
  removedInstances.clear();
}

private void createTypeID(String t, RBSType rbst, Rete r)
     throws JessException {
  Fact f = new Fact("type-identifier", r);
  f.setSlotValue("identifier", new Value(t, RU.STRING));
  f.setSlotValue("value", new Value(rbst));
  r.assert(f);
```

```
}

private void createInstanceID(String i, RBSInstance rbsi, Rete r)
     throws JessException {
  Fact f = new Fact("instance-identifier", r);
  f.setSlotValue("identifier", new Value(i, RU.STRING));
  f.setSlotValue("value", new Value(rbsi));
  r.assert(f);
}

private void removeTypeFact(String t, Rete r)
     throws JessException {
  RBSType type = (RBSType)removedTypes.get(t);

  Fact f = new Fact("type-identifier", r);
  f.setSlotValue("identifier", new Value(t, RU.STRING));
  f.setSlotValue("value", new Value(type));
  r.retract(f);

  type.deleteFacts(r);
}

private void removeInstanceFact(String i, Rete r)
     throws JessException {
  RBSInstance inst = (RBSInstance)removedInstances.get(i);

  Fact f = new Fact("instance-identifier", r);
  f.setSlotValue("identifier", new Value(i, RU.STRING));
  f.setSlotValue("value", new Value(inst));
  r.retract(f);

  inst.deleteFacts(r);
}

/* Modify certain Factory methods */

public void addType(String name, Type type)
     throws TypeExistsException {
  addedTypes.addElement(name);
  super.addType(name, type);
}

public void removeType(String name) {
  if (addedTypes.contains(name)) addedTypes.removeElement(name);
  else {
    try {
      removedTypes.put(name, super.getType(name));
      super.removeType(name);
    }
    catch (NoSuchTypeException nste) { /* doesn't matter */ }
  }
}

public void addInstance(String name, Instance instance)
     throws InstanceExistsException {
  addedInstances.addElement(name);
  super.addInstance(name, instance);
}
```

```
  public void removeInstance(String name) {
    if (addedInstances.contains(name)) addedInstances.removeElement(name);
    else {
      try {
        removedInstances.put(name, super.getInstance(name));
        super.removeInstance(name);
      }
      catch (NoSuchInstanceException nsie) { /* doesn't matter */ }
    }
  }

}
```

8.2.4 *core/Type.java*

```
/* Copyright (c) 1999 Jennifer E. Grucza
   All Rights Reserved. */

package DevelopmentConsultant.core;

import java.util.*;

public class Type implements java.io.Serializable {

  /** A Type acts as a sort of subset of a Factory, but unlike
      a Factory, it specifies properties that Instances of
      this Type hold.  For example, one could have a Factory
      of Users, one of whose Types is programmer.  The properties
      a programmer has would include what programming languages
      they know, etc.  **/

  private Vector properties = new Vector();
  private Hashtable factoryChoices = new Hashtable();

  public Vector getProperties() {
    /** Returns all the properties held by this Type **/
    return properties;
  }

  public void addProperty(String property, String factory)
       throws PropertyExistsException {

    /** Add a new property to this Type with the name <property>.
        This values for this property will be chosen from the
        instances of <factory>.  If <factory> is "none", the value
        is entered as a String.  (There remains the possibility
        to later expand this to other types of Objects - perhaps
        one could include images as property values.)  **/

    if (factoryChoices.containsKey(property)) {
      throw new PropertyExistsException();
    }
    else {
      properties.addElement(property);
      factoryChoices.put(property, factory);
    }
  }

  public String getPropertyFactory(String property)
```

```
        throws NoSuchPropertyException {

    /** Returns the String representation of the Factory from
        which the property <property>'s value is chosen.  **/

    if (!factoryChoices.containsKey(property)) {
      throw new NoSuchPropertyException();
    }
    else {
      return (String)factoryChoices.get(property);
    }
  }

  public void removeProperty(String property) {
    properties.removeElement(property);
    factoryChoices.remove(property);
  }

  public Enumeration properties() {
    return properties.elements();
  }
}
```

8.2.5 *core/RBSType.java*

```
/* Copyright (c) 1999 Jennifer E. Grucza
   All Rights Reserved. */

package DevelopmentConsultant.core;

import DevelopmentConsultant.rules.*;
import java.util.*;
import jess.*;

public class RBSType extends Type implements RBSObject {

  /** At this time, no additional information is needed
      from Types, but future expansion might make it
      necessary.  **/

  public void createFacts(Rete r) {}
  public void updateFacts(Rete r) {}
  public void deleteFacts(Rete r) {}
}
```

8.2.6 *core/Instance.java*

```
/* Copyright (c) 1999 Jennifer E. Grucza
   All Rights Reserved. */

package DevelopmentConsultant.core;

import java.util.*;

public class Instance implements java.io.Serializable {
```

```
/** An Instance belongs to a Factory, and can be of one or more
    Types.  It inherits the properties from all of the Types
    it belongs to.  **/

protected Hashtable properties = new Hashtable();
private Vector types = new Vector();

public Vector getTypes() {
  /** Return all Types this Instance belongs to.  **/
  return types;
}

public void addType(String t) throws TypeExistsException {
  /** Add a new Type to those this Instance has.  **/
  if (types.contains(t)) {
    throw new TypeExistsException();
  }
  else {
    types.addElement(t);
  }
}

public String getPropertyFactory(String property, Factory f)
     throws NoSuchTypeException {
  /** Non-deterministically chooses a type <t> in this.types such that
      <t> contains the property <property>, and returns the String name
      of the Factory from whence the property value can be chosen
      (or "none" if none).  **/

  Enumeration enum = types.elements();
  while (enum.hasMoreElements()) {
    Type t = f.getType((String)enum.nextElement());
    try {
      return t.getPropertyFactory(property);
    }
    catch (NoSuchPropertyException nspe) {
      continue;
    }
  }
  throw new NoSuchTypeException();
}

public void removeType(String t) {
  /** Remove a Type from this Instance's list of Types.  Note:
      this does not remove any property values which are associated
      with the removed type.  **/
  types.removeElement(t);
}

public Enumeration types() {
  return types.elements();
}

public Object getPropertyValue(String property)
     throws NoSuchPropertyException {
  if (properties.containsKey(property)) {
    return properties.get(property);
  }
  else {
```

```
      throw new NoSuchPropertyException();
    }
  }

  public void putPropertyValue(String property, Object value) {
    properties.put(property, value);
  }

  public String toString() {
    StringBuffer s = new StringBuffer();
    s.append("\t\ttypes: ");
    Enumeration enum = types();
    while (enum.hasMoreElements()) {
      s.append((String)enum.nextElement() + ". ");
    }
    s.append("\n\t\tproperties:\n");
    enum = properties.keys();
    while (enum.hasMoreElements()) {
      String key = (String)enum.nextElement();
      try {
        Object value = getPropertyValue(key);
        if (value instanceof String) {
          s.append("\t\t\t" + key + "\t" + (String)value + "\n");
        }
        else if (value instanceof Vector) {
          s.append("\t\t\t" + key + "\t" +
           DevelopmentConsultant.gui.Utilities.
               vectorToString((Vector)value) + "\n");
        }
        else {
          s.append("\t\t\t" + key + "\tUnknown value type");
        }
      }
      catch (NoSuchPropertyException nspe) {
        s.append("\t\t\t" + key + "\tProperty value was unobtainable");
      }
    }
    return s.toString();
  }
}
```

8.2.7 *core/RBSInstance.java*

```
/* Copyright (c) 1999 Jennifer E. Grucza
   All Rights Reserved. */

package DevelopmentConsultant.core;

import DevelopmentConsultant.rules.*;
import java.util.*;
import jess.*;

public class RBSInstance extends Instance implements RBSObject {

  private boolean dirtyBit = true;
  private Vector addedTypes = new Vector();
  private Vector removedTypes = new Vector();
  private Vector changedProperties = new Vector();
```

```
public void createFacts(Rete r) throws JessException {
  Enumeration enum = super.types();
  while (enum.hasMoreElements()) {
    typeFact((String)enum.nextElement(), r);
  }
  enum = super.properties.keys();
  while (enum.hasMoreElements()) {
    propertyValueFact((String)enum.nextElement(), r);
  }
  dirtyBit = false;
  addedTypes.removeAllElements();
  removedTypes.removeAllElements();
  changedProperties.removeAllElements();
}

public void updateFacts(Rete r) throws JessException {
  if (!dirtyBit) return;
  Enumeration enum = addedTypes.elements();
  while (enum.hasMoreElements()) {
    typeFact((String)enum.nextElement(), r);
  }
  enum = removedTypes.elements();
  while (enum.hasMoreElements()) {
    removeTypeFact((String)enum.nextElement(), r);
  }
  enum = changedProperties.elements();
  while (enum.hasMoreElements()) {
    propertyValueFact((String)enum.nextElement(), r);
  }
  dirtyBit = false;
  addedTypes.removeAllElements();
  removedTypes.removeAllElements();
  changedProperties.removeAllElements();
}

public void deleteFacts(Rete r) throws JessException {
  Enumeration enum = super.types();
  while (enum.hasMoreElements()) {
    removeTypeFact((String)enum.nextElement(), r);
  }
  dirtyBit = false;
  addedTypes.removeAllElements();
  removedTypes.removeAllElements();
  changedProperties.removeAllElements();
}

private void typeFact(String type, Rete r) throws JessException {
  Fact f = new Fact("instance-has-type", r);
  f.setSlotValue("instance", new Value(this));
  f.setSlotValue("type", new Value(type, RU.STRING));
  r.assert(f);
}

private void removeTypeFact(String type, Rete r) throws JessException {
  Fact f = new Fact("instance-has-type", r);
  f.setSlotValue("instance", new Value(this));
  f.setSlotValue("type", new Value(type, RU.STRING));
  r.retract(f);
```

```
  }

  private void propertyValueFact(String property, Rete r)
       throws JessException {
    try {
      Object obj = super.getPropertyValue(property);
      if (obj instanceof String) {
        Fact f = new Fact("instance-has-property-value", r);
        f.setSlotValue("instance", new Value(this));
        f.setSlotValue("property", new Value(property, RU.STRING));
        f.setSlotValue("value", new Value((String)obj, RU.STRING));
        r.assert(f);
      }
      else if (obj instanceof Vector) {
        Enumeration enum = ((Vector)obj).elements();
        while (enum.hasMoreElements()) {
          Object obj2 = enum.nextElement();
          if (obj2 instanceof String) {
            Fact f = new Fact("instance-has-property-value", r);
            f.setSlotValue("instance", new Value(this));
            f.setSlotValue("property", new Value(property, RU.STRING));
            f.setSlotValue("value", new Value((String)obj2, RU.STRING));
            r.assert(f);
          }
        }
      }
    }
    catch (NoSuchPropertyException nspe) {
      System.out.println("Internal Error: " + nspe.toString());
    }
  }

  /* Modify some of Instance's inherited methods */

  public void addType(String t) throws TypeExistsException {
    if (removedTypes.contains(t)) removedTypes.removeElement(t);
    else {
      addedTypes.addElement(t);
      super.addType(t);
    }
    dirtyBit = true;
  }

  public void removeType(String t) {
    if (addedTypes.contains(t)) addedTypes.removeElement(t);
    else {
      removedTypes.removeElement(t);
      super.removeType(t);
    }
    dirtyBit = true;
  }

  public void putPropertyValue(String property, Object value) {
    changedProperties.addElement(property);
    super.putPropertyValue(property, value);
    dirtyBit = true;
  }
}
```

8.3 Rule-based system

8.3.1 *rules/RBSObject.java*

```
/* Copyright (c) 1999 Jennifer E. Grucza
   All Rights Reserved. */

package DevelopmentConsultant.rules;

import java.util.*;
import jess.*;

public interface RBSObject {

  /** An interface for objects which hold knowledge which
      is used by the rule-based system.  **/

  void createFacts(Rete r) throws JessException;
  /** Using the jess rule-based system shell (which is based
      on NASA's CLIPS), the Object creates any pertinent
      facts about itself and addes them to the Rete engine
      <r>.  **/

  void updateFacts(Rete r) throws JessException;
  /** Updates changed information.  **/

  void deleteFacts(Rete r) throws JessException;
  /** Deletes removed information.  **/
}
```

8.3.2 *rules/RBSEngine.java*

```
/* Copyright (c) 1999 Jennifer E. Grucza
   All Rights Reserved. */

package DevelopmentConsultant.rules;

import DevelopmentConsultant.core.*;
import jess.*;
import java.util.*;

public class RBSEngine extends Rete {

  /** This class is the rule-based engine used for applications
      such as bug-reporting.  **/

  public RBSEngine() throws JessException {
    super.setFactDuplication(false);
    setupTemplates();
  }

  private void setupTemplates() throws JessException {
    /** Sets up templates we will need.  **/

    Value empty = new Value("");
```

```
    Deftemplate dt = new Deftemplate("instance-has-type",
                                     "The types an instance belongs to");
    dt.addSlot("instance", empty, "ANY");
    dt.addSlot("type", empty, "STRING");
    super.addDeftemplate(dt);

    dt = new Deftemplate("instance-has-property-value",
                         "The value for a given instance and property");
    dt.addSlot("instance", empty, "ANY");
    dt.addSlot("property", empty, "STRING");
    dt.addSlot("value", empty, "ANY");
    super.addDeftemplate(dt);

    dt = new Deftemplate("type-identifier",
                         "The identifying String for a Type value");
    dt.addSlot("identifier", empty, "STRING");
    dt.addSlot("value", empty, "ANY");
    super.addDeftemplate(dt);

    dt = new Deftemplate("instance-identifier",
                         "The identifying String for an Instance value");
    dt.addSlot("identifier", empty, "STRING");
    dt.addSlot("value", empty, "ANY");
    super.addDeftemplate(dt);
  }

  public void loadFacts(Data data) throws JessException {
    /** Loads in all facts from <data> **/

    Enumeration enum = data.factories.elements();
    while (enum.hasMoreElements()) {
      RBSFactory f = (RBSFactory)enum.nextElement();
      f.createFacts(this);
    }
  }
}
```

8.4 Graphical user interface

8.4.1 *gui/Main.java*

```
/* Copyright (c) 1999 Jennifer E. Grucza
   All Rights Reserved. */

package DevelopmentConsultant.gui;

import javax.swing.*;
import java.awt.*;

public class Main {

  /** This class starts the program.  **/

  public static void main(String[] args) {
```

```
    ProgramDesktop d = new ProgramDesktop("DevelopmentConsultant");
    d.setLocation(200, 200);
    d.setVisible(true);
  }
}
```

8.4.2 *gui/ProgramDesktop.java*

```
/* Copyright (c) 1999 Jennifer E. Grucza
   All Rights Reserved. */

package DevelopmentConsultant.gui;

import DevelopmentConsultant.GlobalVars;
import DevelopmentConsultant.core.*;
import DevelopmentConsultant.rules.*;
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import java.util.*;
import java.io.*;
import jess.*;

public class ProgramDesktop extends JFrame {

  /** Desktop area onto which all other windows fall.  **/

  private Data data = new Data();
  private JDesktopPane desk;
  private Hashtable frames = new Hashtable();
  private int frameCount = 0;
  private ReportBugAction bugAction;

  public ProgramDesktop(String title) {
    super(title);

    addWindowListener(new BasicWindowMonitor());

    desk = new JDesktopPane();
    desk.setPreferredSize(new Dimension(900, 700));
    setContentPane(desk);

    createMenuBar();

    pack();
  }

  private void createMenuBar() {
    JMenuBar mb = new JMenuBar();
    setJMenuBar(mb);

    JMenu menu = new JMenu("File");
    menu.setMnemonic('F');
    JMenuItem item;
    item = menu.add(new LoadAction(GlobalVars.saveFilename));
    item.setMnemonic('L');
    item = menu.add(new SaveAction(GlobalVars.saveFilename));
    item.setAccelerator(KeyStroke.getKeyStroke('S',
```

```
                                        java.awt.Event.CTRL_MASK,
                                        false));
    item.setMnemonic('S');
    menu.addSeparator();
    item = menu.add(new QuitAction());
    item.setMnemonic('X');
    mb.add(menu);

    menu = new JMenu("Edit");
    menu.setMnemonic('E');
    JMenu submenu = new JMenu("Types");
    submenu.setMnemonic('T');
    Enumeration enum = data.factories.keys();
    while (enum.hasMoreElements()) {
      submenu.add(new TypesFrameAction((String)enum.nextElement()));
    }
    menu.add(submenu);

    submenu = new JMenu("Instances");
    submenu.setMnemonic('I');
    enum = data.factories.keys();
    while (enum.hasMoreElements()) {
      submenu.add(new InstancesFrameAction((String)enum.nextElement()));
    }
    menu.add(submenu);

    item = menu.add(new RuleEditFrameAction("Rules"));
    item.setMnemonic('R');

    mb.add(menu);

    menu = new JMenu("Bug Tracking");
    menu.setMnemonic('B');
    bugAction = new ReportBugAction(data, this);
    item = menu.add(bugAction);
    item.setMnemonic('R');
    item = menu.add(new BugsFrameAction());
    item.setMnemonic('V');
    mb.add(menu);
  }

  public void update() {
    /** Go through each internal frame and update its contents.  **/

    JInternalFrame[] frames = desk.getAllFrames();
    for (int j = 0; j < frames.length; j++) {
      JInternalFrame jif = frames[j];
      Updateable u = (Updateable)jif;
      u.updateContents(data);
    }
    bugAction.updateContents(data);
  }

  class TypesFrameAction extends AbstractAction {
    public TypesFrameAction(String name) {
      super(name);
    }

    public void actionPerformed(ActionEvent e) {
```

```
    String name = e.getActionCommand();
    String key = "Edit Types of " + name;
    if (frames.containsKey(key)) {
      if (((JInternalFrame)frames.get(key)).isClosed()) {
        JInternalFrame f = new EditTypesInternalFrame(name, data);
        f.setBounds(frameCount * 30, frameCount * 30, 400, 300);
        frameCount++;
        frames.put(key, f);
        desk.add(f);
        desk.moveToFront(f);
      }
      else {
        desk.moveToFront((JInternalFrame)frames.get(key));
      }
    }
    else {
      JInternalFrame f = new EditTypesInternalFrame(name, data);
      f.setBounds(frameCount * 30, frameCount * 30, 400, 300);
      frameCount++;
      frames.put(key, f);
      desk.add(f);
      desk.moveToFront(f);
    }
  }
}

class InstancesFrameAction extends AbstractAction {
  public InstancesFrameAction(String name) {
    super(name);
  }

  public void actionPerformed(ActionEvent e) {
    String name = e.getActionCommand();
    String key = "Edit " + name;
    if (frames.containsKey(key)) {
      if (((JInternalFrame)frames.get(key)).isClosed()) {
        JInternalFrame f = new EditInstancesInternalFrame(name, data);
        f.setBounds(frameCount * 30, frameCount * 30, 600, 500);
        frameCount++;
        frames.put(key, f);
        desk.add(f);
        desk.moveToFront(f);
      }
      else {
        desk.moveToFront((JInternalFrame)frames.get(key));
      }
    }
    else {
      JInternalFrame f = new EditInstancesInternalFrame(name, data);
      f.setBounds(frameCount * 30, frameCount * 30, 600, 500);
      frameCount++;
      frames.put(key, f);
      desk.add(f);
      desk.moveToFront(f);
    }
  }
}

class RuleEditFrameAction extends AbstractAction {
```

```
  public RuleEditFrameAction(String s) {
    super(s);
  }

  public void actionPerformed(ActionEvent e) {
    String name = e.getActionCommand();
    if (frames.containsKey(name)) {
      if (((JInternalFrame)frames.get(name)).isClosed()) {
        JInternalFrame f = new EditRulesInternalFrame(data);
        f.setBounds(frameCount * 30, frameCount * 30, 700, 400);
        frameCount++;
        frames.put(name, f);
        desk.add(f);
        desk.moveToFront(f);
      }
      else {
        desk.moveToFront((JInternalFrame)frames.get(name));
      }
    }
    else {
      JInternalFrame f = new EditRulesInternalFrame(data);
      f.setBounds(frameCount * 30, frameCount * 30, 700, 400);
      frameCount++;
      frames.put(name, f);
      desk.add(f);
      desk.moveToFront(f);
    }
  }
}


class BugsFrameAction extends AbstractAction {
  public BugsFrameAction() {
    super("View Bug Reports");
  }

  public void actionPerformed(ActionEvent e) {
    String name = e.getActionCommand();
    if (frames.containsKey(name)) {
      if (((JInternalFrame)frames.get(name)).isClosed()) {
        JInternalFrame f = new ViewBugsInternalFrame(name, data);
        f.setBounds(frameCount * 30, frameCount * 30, 800, 600);
        frameCount++;
        frames.put(name, f);
        desk.add(f);
        desk.moveToFront(f);
      }
      else {
        desk.moveToFront((JInternalFrame)frames.get(name));
      }
    }
    else {
      JInternalFrame f = new ViewBugsInternalFrame(name, data);
      f.setBounds(frameCount * 30, frameCount * 30, 800, 600);
      frameCount++;
      frames.put(name, f);
      desk.add(f);
      desk.moveToFront(f);
    }
```

```
    }
  }

  class LoadAction extends AbstractAction {

    String filename;

    public LoadAction(String filename) {
      super("Load");
      this.filename = filename;
    }

    public void actionPerformed(ActionEvent e) {
      int i = JOptionPane.showConfirmDialog(ProgramDesktop.this,
state.  Continue?",                                "Loading now will overwrite current
                                                   "Warning",
                                                   JOptionPane.YES_NO_OPTION,
                                                   JOptionPane.WARNING_MESSAGE);
      if (i == JOptionPane.NO_OPTION) return;

      try {
        FileInputStream fileIn = new FileInputStream(filename);
        ObjectInputStream in = new ObjectInputStream(fileIn);
        data = (Data)in.readObject();
        in.close();
        fileIn.close();

        update();
      }
      catch (ClassNotFoundException cnfe) {
        GUIUtils.errorDialog(ProgramDesktop.this, cnfe);
      }
      catch (IOException ioe) {
        GUIUtils.errorDialog(ProgramDesktop.this, ioe);
      }
    }
  }

  public class SaveAction extends AbstractAction {

    String filename;

    public SaveAction(String filename) {
      super("Save");
      this.filename = filename;
    }

    public void actionPerformed(ActionEvent e) {
      try {
        int i = JOptionPane.showConfirmDialog(ProgramDesktop.this,
                                              "Save state?",
                                               "Confirm",
                                               JOptionPane.YES_NO_OPTION);
        if (i == JOptionPane.NO_OPTION) return;
        FileOutputStream fileOut = new FileOutputStream(filename);
        ObjectOutputStream out = new ObjectOutputStream(fileOut);
        out.writeObject(data);
        out.close();
```

```
        fileOut.close();

        FileWriter file =
          new FileWriter("/mit/jgrucza/DevelopmentConsultant/toString");
        file.write(data.toString());
        file.close();
      }
      catch (IOException ioe) {
        GUIUtils.errorDialog(ProgramDesktop.this, ioe);
      }
    }
  }

  class QuitAction extends AbstractAction {
    public QuitAction() {
      super("Exit");
    }

    public void actionPerformed(ActionEvent e) {
      System.exit(0);
    }
  }
}
```

8.4.3 *gui/BasicWindowMonitor.java*

```
/* From the O'Reilly Java Swing book by Eckstein, Loy, & Wood, 1998 */

package DevelopmentConsultant.gui;

import java.awt.event.*;
import java.awt.Window;

public class BasicWindowMonitor extends WindowAdapter {
  public void windowClosing(WindowEvent e) {
    Window w = e.getWindow();
    w.setVisible(false);
    w.dispose();
    System.exit(0);
  }
}
```

8.4.4 *gui/GUIUtils.java*

```
/* Copyright (c) 1999 Jennifer E. Grucza
   All Rights Reserved. */

package DevelopmentConsultant.gui;

import javax.swing.*;
import javax.swing.event.*;
import java.awt.*;
import java.awt.event.*;

public class GUIUtils {
```

```
  /** Provides a couple of methods for displaying error or warning
      messages using dialog boxes.  **/

  public static void errorDialog(Component f, Exception e) {
    JTextArea ta = new JTextArea(e.toString());
    ta.setEditable(false);
    JOptionPane.showMessageDialog(f,
                                  new JScrollPane(ta),
                                  "Internal Error",
                                  JOptionPane.ERROR_MESSAGE);
  }

  public static void warningDialog(Component f, String s) {
    JOptionPane.showMessageDialog(f,
                                  s,
                                  "Warning",
                                  JOptionPane.WARNING_MESSAGE);
  }

}
```

8.4.5 *gui/Updateable.java*

```
/* Copyright (c) 1999 Jennifer E. Grucza
   All Rights Reserved. */

package DevelopmentConsultant.gui;

import DevelopmentConsultant.core.*;
import javax.swing.*;

public interface Updateable {

  /** Classes implementing Updateable update their displays
      using the information contained in <data>.  **/

  public void updateContents(Data data);
}
```

8.4.6 *gui/Utilities.java*

```
/* Copyright (c) 1999 Jennifer E. Grucza
   All Rights Reserved. */

package DevelopmentConsultant.gui;

import java.util.*;
import javax.swing.*;

public class Utilities {

  /** Assorted miscellaneous utilities needed by other gui classes.  **/

  public static Vector convert(Enumeration enum) {
    /** Takes elements in an Enumeration and inserts them
        into a Vector.  **/
```

```
    Vector out = new Vector();
    while (enum.hasMoreElements()) {
      out.addElement(enum.nextElement());
    }
    return out;
  }

  public static DefaultListModel vector2DLM(Vector v) {
    /** Creates a DefaultListModel from the contents of Vector <v>.  **/

    Enumeration enum = v.elements();
    DefaultListModel dlm = new DefaultListModel();
    while (enum.hasMoreElements()) {
      dlm.addElement(enum.nextElement());
    }
    return dlm;
  }

  public static DefaultListModel complement(Vector whole, Vector firstPart) {
    /** Creates and returns a DefaultListModel with all the elements in
        <whole> which are not also in <firstPart> **/

    Collections.sort(whole);
    DefaultListModel rest = new DefaultListModel();
    Enumeration enum = whole.elements();
    while (enum.hasMoreElements()) {
      Object element = enum.nextElement();
      if (!firstPart.contains(element)) {
        rest.addElement(element);
      }
    }
    return rest;
  }

  public static String vectorToString(Vector v) {
    /** Returns a simple String representation of a Vector.  **/

    StringBuffer s = new StringBuffer();
    Enumeration enum = v.elements();
    while (enum.hasMoreElements()) {
      s.append((String)enum.nextElement() + ". ");
    }
    return s.toString();
  }
}
```

8.4.7 *gui/EditTypesInternalFrame.java*

```
/* Copyright (c) 1999 Jennifer E. Grucza
   All Rights Reserved. */

package DevelopmentConsultant.gui;

import DevelopmentConsultant.core.*;
import java.util.*;
import javax.swing.*;
```

```
import javax.swing.event.*;
import java.awt.*;
import java.awt.event.*;

public class EditTypesInternalFrame extends JInternalFrame
                                    implements Updateable {

  private String name;
  private Data data;
  private Factory factory;
  private JList types;
  private JList properties;
  private String selectedType;
  private String selectedProperty;
  private JButton trb; // type remove button
  private JButton pnb; // property new button
  private JButton prb; // property remove button

  public EditTypesInternalFrame(String name, Data data) {
    super("Edit Types of " + name, true, true, true, true);
    this.name = name;
    this.data = data;
    factory = (Factory)data.factories.get(name);

    /* Left side contains names of types. */

    JPanel p1 = new JPanel(new BorderLayout());
    types = new JList(factory.getTypes());
    types.setSelectionMode(ListSelectionModel.SINGLE_SELECTION);
    types.addListSelectionListener(new TypeListener());
    JScrollPane scroll = new JScrollPane(types);
    scroll.setPreferredSize(new Dimension(100, 220));
    p1.add(scroll, BorderLayout.CENTER);

    JPanel buttonP = new JPanel();
    JButton b = new JButton("New");
    b.setActionCommand("New type");
    b.addActionListener(new ButtonListener());
    buttonP.add(b);

    trb = new JButton("Remove");
    trb.setEnabled(false);
    trb.setActionCommand("Remove type");
    trb.addActionListener(new ButtonListener());
    buttonP.add(trb);
    p1.add(buttonP, BorderLayout.SOUTH);

    /* Right side contains names of properties for selected type. */

    JPanel p2 = new JPanel(new BorderLayout());
    properties = new JList();
    properties.setSelectionMode(ListSelectionModel.SINGLE_SELECTION);
    properties.addListSelectionListener(new PropertyListener());
    scroll = new JScrollPane(properties);
    scroll.setPreferredSize(new Dimension(100, 220));
    p2.add(scroll, BorderLayout.CENTER);

    buttonP = new JPanel();
    pnb = new JButton("New");
```

```
    pnb.setEnabled(false);
    pnb.setActionCommand("New property");
    pnb.addActionListener(new ButtonListener());
    buttonP.add(pnb);

    prb = new JButton("Remove");
    prb.setEnabled(false);
    prb.setActionCommand("Remove property");
    prb.addActionListener(new ButtonListener());
    buttonP.add(prb);
    p2.add(buttonP, BorderLayout.SOUTH);

    p1.setPreferredSize(new Dimension(180, 500));
    p2.setPreferredSize(new Dimension(180, 500));

    JSplitPane sp = new JSplitPane(JSplitPane.HORIZONTAL_SPLIT, p1, p2);
    getContentPane().add(sp, BorderLayout.CENTER);
  }

  class TypeListener implements ListSelectionListener {
    public void valueChanged(ListSelectionEvent e) {
      selectedType = (String)types.getSelectedValue();
      if (selectedType != null) {
        trb.setEnabled(true);
        pnb.setEnabled(true);
        try {
          Vector p = factory.getType(selectedType).getProperties();
          properties.setListData(p);
        }
        catch (NoSuchTypeException nste) {
          // shouldn't happen, but in case...
          GUIUtils.errorDialog(EditTypesInternalFrame.this, nste);
          return;
        }
      }
      else {
        trb.setEnabled(false);
        pnb.setEnabled(false);
        properties.setListData(new Vector());
      }
    }
  }

  class PropertyListener implements ListSelectionListener {
    public void valueChanged(ListSelectionEvent e) {
      selectedProperty = (String)properties.getSelectedValue();
      if (selectedProperty != null) prb.setEnabled(true);
      else prb.setEnabled(false);
    }
  }

  class ButtonListener implements ActionListener {
    public void actionPerformed(ActionEvent e) {
      if (e.getActionCommand() == "New type") {
        String s = JOptionPane.showInputDialog(EditTypesInternalFrame.this,
                                               "Enter new type name");
        if (s != null) {
          try {
            factory.addType(s, new RBSType());
```

```
        }
        catch (TypeExistsException tee) {
          GUIUtils.warningDialog(EditTypesInternalFrame.this,
                       "Type by that name already exists");
          return;
        }
        Vector v = factory.getTypes();
        Collections.sort(v);
        types.setListData(v);
      }
    }
    else if (e.getActionCommand() == "Remove type") {
      int i = JOptionPane.showConfirmDialog(EditTypesInternalFrame.this,
                                            "Remove this type?");
      if (i == JOptionPane.YES_OPTION) {
        factory.removeType(selectedType);
        factory.removeType(selectedType); // odd unexplained bug
        Vector v = factory.getTypes();
        Collections.sort(v);
        types.setListData(v);
        properties.setListData(new Vector());
        trb.setEnabled(false);
      }
    }
    else if (e.getActionCommand() == "New property") {
      String s = JOptionPane.showInputDialog(EditTypesInternalFrame.this,
                                    "Enter new property name");
      if (s != null) {
        Type t;
        try {
          t = factory.getType(selectedType);
        }
        catch (NoSuchTypeException nste) {
          // shouldn't happen...
          GUIUtils.errorDialog(EditTypesInternalFrame.this, nste);
          return;
        }

        Object[] choices = new Object[data.factories.size() + 1];
        choices[0] = "none";
        Enumeration enum = data.factories.keys();
        int i = 1;
        while (enum.hasMoreElements()) {
          choices[i] = enum.nextElement();
          i++;
        }
        String s2 = (String)
          JOptionPane.showInputDialog(EditTypesInternalFrame.this,
collection?",                                  "Property value chosen from which
                                               "Input",
                                               JOptionPane.QUESTION_MESSAGE,
                                               null,
                                               choices,
                                               "none");
        if (s2 != null) {
          try {
            t.addProperty(s, s2);
          }
```

```
            catch (PropertyExistsException pee) {
              // shouldn't ever get this, but just in case...
              GUIUtils.errorDialog(EditTypesInternalFrame.this, pee);
              return;
            }
          }
          properties.setListData(t.getProperties());
        }
      }
      else if (e.getActionCommand() == "Remove property") {
        int i = JOptionPane.showConfirmDialog(EditTypesInternalFrame.this,
                                              "Remove this property?");
        if (i == JOptionPane.YES_OPTION) {
          try {
            Type t = factory.getType(selectedType);
            t.removeProperty(selectedProperty);
            properties.setListData(t.getProperties());
          }
          catch (NoSuchTypeException nste) {
            GUIUtils.errorDialog(EditTypesInternalFrame.this, nste);
          }
        }
      }
    }

  public void updateContents(Data data) {
    this.data = data;
    factory = (Factory)data.factories.get(name);

    Vector v = factory.getTypes();
    Collections.sort(v);
    types.setListData(v);
    properties.setListData(new Vector());
  }
}
```

8.4.8 *gui/EditInstancesInternalFrame.java*

```
/* Copyright (c) 1999 Jennifer E. Grucza
   All Rights Reserved. */

package DevelopmentConsultant.gui;

import DevelopmentConsultant.core.*;
import java.util.*;
import javax.swing.*;
import javax.swing.event.*;
import java.awt.*;
import java.awt.event.*;
import jess.*;

public class EditInstancesInternalFrame extends JInternalFrame
                                        implements Updateable {

  private String name;
  private Data data;
  private Factory factory;
```

```
private JList instances;
private JList types;
private JList properties;

private JButton rib;  // remove instance button
private JButton ntb;  // new type button
private JButton rtb;  // remove type button

private JPanel bottomRight;
private CardLayout cardLayout;
private JTextField propertyValue = new JTextField("", 20);
private JList chosen;
private JList choices;

private String selectedInstance;
private String selectedType;
private String selectedProperty;

public EditInstancesInternalFrame(String name, Data data) {
  super("Edit " + name, true, true, true, true);
  this.name = name;
  this.data = data;
  factory = (Factory)data.factories.get(name);

  JPanel topLeft = setupTopLeft();
  JPanel topRight = setupTopRight();

  topLeft.setPreferredSize(new Dimension(350, 220));
  topRight.setPreferredSize(new Dimension(200, 220));

  JSplitPane top = new JSplitPane(JSplitPane.HORIZONTAL_SPLIT,
                                  topLeft,
                                  topRight);

  JScrollPane propertyScroll = setupBottomLeft();
  bottomRight = setupBottomRight();

  propertyScroll.setPreferredSize(new Dimension(180, 200));
  bottomRight.setPreferredSize(new Dimension(370, 200));

  JPanel bottom = new JPanel(new BorderLayout());
  JLabel label = new JLabel("Property Values");
  bottom.add(label, BorderLayout.NORTH);

  JSplitPane bottomMain = new JSplitPane(JSplitPane.HORIZONTAL_SPLIT,
                                         propertyScroll, bottomRight);
  bottom.add(bottomMain, BorderLayout.CENTER);

  JSplitPane whole = new JSplitPane(JSplitPane.VERTICAL_SPLIT,
                                    top, bottom);
  getContentPane().add(whole, BorderLayout.CENTER);
}

private JPanel setupTopLeft() {
  /* Top pane contains list of instances (e.g. users) and list of
     types that instance belongs to (e.g. Basic User, Software Engineer) */

  JPanel topLeft = new JPanel(new BorderLayout());
```

```
    Vector v = factory.getInstances();
    Collections.sort(v);
    instances = new JList(Utilities.vector2DLM(v));
    instances.setSelectionMode(ListSelectionModel.SINGLE_SELECTION);
    instances.addListSelectionListener(new InstanceListener());
    JScrollPane instanceScroll = new JScrollPane(instances);
    topLeft.add(instanceScroll, BorderLayout.CENTER);

    JPanel buttonP = new JPanel();
    JButton nib = new JButton("New");
    nib.setActionCommand("New instance");
    nib.addActionListener(new ButtonListener());
    buttonP.add(nib);
    rib = new JButton("Remove");
    rib.setEnabled(false);
    rib.setActionCommand("Remove instance");
    rib.addActionListener(new ButtonListener());
    buttonP.add(rib);
    topLeft.add(buttonP, BorderLayout.SOUTH);

    return topLeft;
  }

  private JPanel setupTopRight() {
    // list of Types

    JPanel topRight = new JPanel(new BorderLayout());

    JLabel label = new JLabel("Type(s)");
    topRight.add(label, BorderLayout.NORTH);

    types = new JList();
    types.setSelectionMode(ListSelectionModel.SINGLE_SELECTION);
    types.addListSelectionListener(new TypeListener());
    JScrollPane typeScroll = new JScrollPane(types);
    topRight.add(typeScroll, BorderLayout.CENTER);

    JPanel buttonP = new JPanel();
    ntb = new JButton("New");
    ntb.setEnabled(false);
    ntb.setActionCommand("New type");
    ntb.addActionListener(new ButtonListener());
    buttonP.add(ntb);
    rtb = new JButton("Remove");
    rtb.setEnabled(false);
    rtb.setActionCommand("Remove type");
    rtb.addActionListener(new ButtonListener());
    buttonP.add(rtb);
    topRight.add(buttonP, BorderLayout.SOUTH);

    return topRight;
  }

  private JScrollPane setupBottomLeft() {
    /* Bottom pane contains values for each instance's collection
       of properties. */

    properties = new JList();
    properties.setSelectionMode(ListSelectionModel.SINGLE_SELECTION);
```

```
    properties.addListSelectionListener(new PropertiesListener());
    JScrollPane propertyScroll = new JScrollPane(properties);

    return propertyScroll;
  }

  private JPanel setupBottomRight() {
    /* For an Instance <i> and a property <p>, if i.getPropertyFactory(p)
       is "none", let user enter a String value, else provide a way for
       user to choose one or more Instances from that Factory.  **/

    cardLayout = new CardLayout();
    bottomRight = new JPanel(cardLayout);

    bottomRight.add(new JPanel(), "empty");

    JPanel p = new JPanel();
    p.add(propertyValue);
    JButton enter = new JButton("enter");
    enter.addActionListener(new ActionListener() {
      public void actionPerformed(ActionEvent e) {
        try {
          Instance inst = factory.getInstance(selectedInstance);
          inst.putPropertyValue(selectedProperty, propertyValue.getText());
        }
        catch (NoSuchInstanceException nsie) {
          GUIUtils.errorDialog(EditInstancesInternalFrame.this, nsie);
        }
      }
    } );
    p.add(enter);
    bottomRight.add(p, "text field");

    p = new JPanel(new BorderLayout());
    Box upper = new Box(BoxLayout.X_AXIS);
    chosen = new JList(new DefaultListModel());
    JScrollPane sp = new JScrollPane(chosen);
    sp.setPreferredSize(new Dimension(200, 200));
    upper.add(sp);

    Box middle = new Box(BoxLayout.Y_AXIS);
    JButton addB = new JButton("<= add");
    addB.setAlignmentX(Component.CENTER_ALIGNMENT);
    addB.addActionListener(new ActionListener() {
      public void actionPerformed(ActionEvent e) {
        DefaultListModel chosenList = (DefaultListModel)chosen.getModel();
        DefaultListModel choicesList = (DefaultListModel)choices.getModel();
        Object[] selected = choices.getSelectedValues();
        for (int i = 0; i < selected.length; i++) {
          chosenList.addElement(selected[i]);
          choicesList.removeElement(selected[i]);
        }
      }
    });
    middle.add(addB);

    JButton removeB = new JButton("remove =>");
    removeB.setAlignmentX(Component.CENTER_ALIGNMENT);
    removeB.addActionListener(new ActionListener() {
```

```
    public void actionPerformed(ActionEvent e) {
      DefaultListModel chosenList = (DefaultListModel)chosen.getModel();
      DefaultListModel choicesList = (DefaultListModel)choices.getModel();
      Object[] selected = chosen.getSelectedValues();
      for (int i = 0; i < selected.length; i++) {
        choicesList.addElement(selected[i]);
        chosenList.removeElement(selected[i]);
      }
    }
  });
  middle.add(removeB);
  upper.add(middle);

  choices = new JList(new DefaultListModel());
  sp = new JScrollPane(choices);
  sp.setPreferredSize(new Dimension(200, 200));
  upper.add(sp);
  p.add(upper, BorderLayout.CENTER);

  enter = new JButton("enter");
  enter.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
      try {
        Instance inst = (Instance)factory.getInstance(selectedInstance);
        Vector v = new Vector();
        Enumeration enum = ((DefaultListModel)chosen.getModel()).elements();
        while (enum.hasMoreElements()) {
          v.addElement(enum.nextElement());
        }
        inst.putPropertyValue(selectedProperty, v);
      }
      catch (NoSuchInstanceException nsie) {
        GUIUtils.errorDialog(EditInstancesInternalFrame.this, nsie);
      }
    }
  });
  p.add(enter, BorderLayout.SOUTH);

  bottomRight.add(p, "list");
  cardLayout.show(bottomRight, "empty");

  return bottomRight;
}

class InstanceListener implements ListSelectionListener {
  public void valueChanged(ListSelectionEvent e) {
    selectedInstance = (String)instances.getSelectedValue();
    if (selectedInstance != null) {
      rib.setEnabled(true);
      ntb.setEnabled(true);
      try {
        Instance inst = factory.getInstance(selectedInstance);
        types.setListData(inst.getTypes());
        Vector v = combineProperties(inst);
        properties.setListData(v);
      }
      catch (NoSuchInstanceException nsie) {
        // shouldn't happen
        GUIUtils.errorDialog(EditInstancesInternalFrame.this, nsie);
```

```
      }
    }
    else {
      rib.setEnabled(false);
      ntb.setEnabled(false);
      types.setListData(new Vector());
      properties.setListData(new Vector());
      cardLayout.show(bottomRight, "empty");
    }
  }
}

private Vector combineProperties(Instance inst) {
  /* Combine properties from all Types inst belongs to. */

  Enumeration enum = inst.types();
  Vector v = new Vector();
  try {
    while (enum.hasMoreElements()) {
      Enumeration enum2 =
        factory.getType((String)enum.nextElement()).properties();
      while (enum2.hasMoreElements()) {
        Object element = enum2.nextElement();
        if (!v.contains(element)) {
          v.addElement(element);
        }
      }
    }
  }
  catch (NoSuchTypeException nste) {
    GUIUtils.errorDialog(EditInstancesInternalFrame.this, nste);
    return null;
  }
  return v;
}

class TypeListener implements ListSelectionListener {
  public void valueChanged(ListSelectionEvent e) {
    selectedType = (String)types.getSelectedValue();
    if (selectedType != null) {
      rtb.setEnabled(true);
    }
    else {
      rtb.setEnabled(false);
    }
  }
}

class PropertiesListener implements ListSelectionListener {
  public void valueChanged(ListSelectionEvent e) {
    selectedProperty = (String)properties.getSelectedValue();
    if (selectedProperty != null) {
      try {
        Instance inst = factory.getInstance(selectedInstance);
        String choice = inst.getPropertyFactory(selectedProperty, factory);
        if (choice.equals("none")) {
          try {
            propertyValue.setText((String)
                                  inst.getPropertyValue(selectedProperty));
```

```
          }
          catch (NoSuchPropertyException nspe) {
            /* No problem, simply means we haven't entered a value for the
               property yet.  */
            propertyValue.setText("");
          }
          cardLayout.show(bottomRight, "text field");
        }
        else {
          try {
            Vector chosenV = (Vector)inst.getPropertyValue(selectedProperty);
            chosen.setModel(Utilities.vector2DLM(chosenV));

            Factory f = (Factory)data.factories.get(choice);
            choices.setModel(Utilities.complement(f.getInstances(), chosenV));
            cardLayout.show(bottomRight, "list");
          }
          catch (NoSuchPropertyException nspe) {
            chosen.setModel(new DefaultListModel());
            Vector v = ((Factory)data.factories.get(choice)).getInstances();
            choices.setModel(Utilities.vector2DLM(v));
            cardLayout.show(bottomRight, "list");
          }
        }
      }
      catch (NoSuchTypeException nste) {
        GUIUtils.errorDialog(EditInstancesInternalFrame.this, nste);
      }
      catch (NoSuchInstanceException nsie) {
        GUIUtils.errorDialog(EditInstancesInternalFrame.this, nsie);
      }
    }
    else {
      cardLayout.show(bottomRight, "empty");
    }
  }
}

class ButtonListener implements ActionListener {
  public void actionPerformed(ActionEvent e) {
    String command = e.getActionCommand();
    if (command == "New instance") {

      JPanel main = new JPanel(new BorderLayout());
      JPanel nameP = new JPanel();
      nameP.add(new Label("Name"));
      JTextField name = new JTextField(20);
      nameP.add(name);
      main.add(nameP, BorderLayout.NORTH);
      Vector v = factory.getTypes();
      Collections.sort(v);
      JList types = new JList(v);
      main.add(new JScrollPane(types));

      int i = JOptionPane.showConfirmDialog(EditInstancesInternalFrame.this,
                                            main,
                                            "Create new instance",
                                            JOptionPane.OK_CANCEL_OPTION);
```

```
  if (i == JOptionPane.OK_OPTION) {
    Object[] selected = types.getSelectedValues();
    Vector s = new Vector();     // names of types
    for (int j = 0; j < selected.length; j++) {
      s.addElement(selected[j]);
    }
    try {
      Instance inst = new RBSInstance();
      for (int j = 0; j < selected.length; j++) {
        inst.addType((String)selected[j]);
      }
      factory.addInstance(name.getText(), inst);
      v = factory.getInstances();
      Collections.sort(v);
      instances.setModel(Utilities.vector2DLM(v));
      Collections.sort(s);
      types.setListData(s);
    }
    catch (InstanceExistsException iee) {
      GUIUtils.warningDialog(EditInstancesInternalFrame.this,
                             "Instance by that name already exists");
    }
    catch (TypeExistsException tee) {
      GUIUtils.errorDialog(EditInstancesInternalFrame.this, tee);
    }
  }
}
else if (command == "Remove instance") {
  int i = JOptionPane.showConfirmDialog(EditInstancesInternalFrame.this,
                                        "Remove this instance?");
  if (i == JOptionPane.YES_OPTION) {
    DefaultListModel m = (DefaultListModel)instances.getModel();
    m.removeElement(selectedInstance);
    factory.removeInstance(selectedInstance);
    factory.removeInstance(selectedInstance); // odd bug
    EditInstancesInternalFrame.this.types.setListData(new Vector());
    properties.setListData(new Vector());
    rib.setEnabled(false);
  }
}
else if (command == "New type") {
  /* Assemble Vector <choices> so that it contains all the Types
     that the instance doesn't belong to yet. */

  Vector choices = (Vector)factory.getTypes().clone();
  Instance inst;
  try {
    inst = factory.getInstance(selectedInstance);
  }
  catch (NoSuchInstanceException nsie) {
    GUIUtils.errorDialog(EditInstancesInternalFrame.this, nsie);
    return;
  }
  Enumeration enum = inst.types();
  while (enum.hasMoreElements()) {
    choices.remove(enum.nextElement());
  }
  Collections.sort(choices);
```

```
        JList typeChoices = new JList(choices);
        int i = JOptionPane.showConfirmDialog(EditInstancesInternalFrame.this,
                                              new JScrollPane(typeChoices),
                                              "Add type(s)",
                                              JOptionPane.OK_CANCEL_OPTION);

        if (i == JOptionPane.OK_OPTION) {
          Object[] selected = typeChoices.getSelectedValues();
          try {
            for (int j = 0; j < selected.length; j++) {
              inst.addType((String)selected[j]);
            }
            Vector v = inst.getTypes();
            Collections.sort(v);
            types.setListData(v);
            properties.setListData(combineProperties
                                   (factory.getInstance(selectedInstance)));
          }
          catch (TypeExistsException tee) {
            GUIUtils.warningDialog(EditInstancesInternalFrame.this,
                                   "Type by that name already exists");
          }
          catch (NoSuchInstanceException nsie) {
            GUIUtils.errorDialog(EditInstancesInternalFrame.this, nsie);
          }
        }
      }
      else if (command == "Remove type") {
        int i = JOptionPane.showConfirmDialog(EditInstancesInternalFrame.this,
                                              "Remove this type?");
        if (i == JOptionPane.YES_OPTION) {
          try {
            Instance inst = factory.getInstance(selectedInstance);
            inst.removeType(selectedType);
            types.setListData(inst.getTypes());
            properties.setListData(combineProperties
                                   (factory.getInstance(selectedInstance)));
          }
          catch (NoSuchInstanceException nsie) {
            GUIUtils.errorDialog(EditInstancesInternalFrame.this, nsie);
          }
        }
      }
    }
  }

  public void updateContents(Data data) {
    this.data = data;
    factory = (Factory)data.factories.get(name);

    Vector v = factory.getInstances();
    Collections.sort(v);
    instances.setModel(Utilities.vector2DLM(v));
    types.setListData(new Vector());
    properties.setListData(new Vector());
  }

}
```

8.4.9 *gui/EditRulesInternalFrame.java*

```
/* Copyright (c) 1999 Jennifer E. Grucza
   All Rights Reserved. */

package DevelopmentConsultant.gui;

import DevelopmentConsultant.*;
import DevelopmentConsultant.core.*;
import DevelopmentConsultant.rules.*;
import jess.*;
import javax.swing.*;
import javax.swing.event.*;
import java.awt.*;
import java.awt.event.*;
import java.util.*;
import java.io.*;

public class EditRulesInternalFrame extends JInternalFrame
                                    implements Updateable {

  private Data data;
  private JList names;
  private JTextArea rule = new JTextArea(4, 25);
  private String selectedRule;
  private JTextArea helpMessage;
  private JButton save;
  private JButton remove;

  public EditRulesInternalFrame(Data d) {
    super("Edit Rules", true, true, true, true);
    data = d;

    helpMessage = setupHelpMessage();

    names = new JList(Utilities.convert(data.rules.keys()));
    names.setSelectionMode(ListSelectionModel.SINGLE_SELECTION);
    names.addListSelectionListener(new RuleListener());
    JScrollPane scroll = new JScrollPane(names);

    names.setPreferredSize(new Dimension(175, 300));

    rule.setEnabled(false);
    rule.setFont(new Font("Courier", Font.PLAIN, 14));

    JSplitPane sp = new JSplitPane(JSplitPane.HORIZONTAL_SPLIT,
                                   names, new JScrollPane(rule));
    getContentPane().add(sp, BorderLayout.CENTER);

    JPanel buttons = new JPanel();

    JButton newB = new JButton("new");
    newB.addActionListener(new NewListener());
    buttons.add(newB);

    save = new JButton("enter");
    save.addActionListener(new SaveListener());
    save.setEnabled(false);
```

```
    buttons.add(save);

    remove = new JButton("remove");
    remove.addActionListener(new RemoveListener());
    remove.setEnabled(false);
    buttons.add(remove);

    JButton help = new JButton("help");
    help.addActionListener(new HelpListener());
    buttons.add(help);

    getContentPane().add(buttons, BorderLayout.SOUTH);

    pack();
  }

  class RuleListener implements ListSelectionListener {
    public void valueChanged(ListSelectionEvent e) {
      selectedRule = (String)names.getSelectedValue();
      if (selectedRule != null) {
        rule.setEnabled(true);
        rule.setText((String)data.rules.get(selectedRule));
        save.setEnabled(true);
        remove.setEnabled(true);
      }
      else {
        rule.setText("");
        rule.setEnabled(false);
        save.setEnabled(false);
        remove.setEnabled(false);
      }
    }
  }

  class NewListener implements ActionListener {
    public void actionPerformed(ActionEvent e) {
      String s = JOptionPane.showInputDialog(EditRulesInternalFrame.this,
                                             "Enter new rule name (no spaces)");
      if (s != null) {
       if (data.rules.containsKey(s)) {
         GUIUtils.warningDialog(EditRulesInternalFrame.this,
                                "Rule by that name already exists");
         return;
       }
       data.rules.put(s,
                      "(defrule " + s + "\n  (\t\t)\n  =>\n  (assert (\t\t)))");
       names.setListData(Utilities.convert(data.rules.keys()));
      }
    }
  }

  class SaveListener implements ActionListener {
    public void actionPerformed(ActionEvent e) {
      data.rules.put(selectedRule, rule.getText());
    }
  }

  class RemoveListener implements ActionListener {
    public void actionPerformed(ActionEvent e) {
```

```
      int i = JOptionPane.showConfirmDialog(EditRulesInternalFrame.this,
                                            "Remove this rule?");
      if (i == JOptionPane.YES_OPTION) {
        if (data.rules.containsKey(selectedRule)) {
          data.rules.remove(selectedRule);
          updateContents(data);
        }
        else {
          names.setListData(Utilities.convert(data.rules.keys()));
        }
      }
    }
  }

  class HelpListener implements ActionListener {
    public void actionPerformed(ActionEvent e) {
      JScrollPane sp = new JScrollPane(helpMessage);
      sp.setPreferredSize(new Dimension(500, 300));
      JOptionPane.showMessageDialog(EditRulesInternalFrame.this,
                                    sp,
                                    "Rules Help",
                                    JOptionPane.INFORMATION_MESSAGE);
    }
  }

  private JTextArea setupHelpMessage() {
    try {
      FileReader fileIn = new FileReader(GlobalVars.rulesHelpFilename);
      BufferedReader in = new BufferedReader(fileIn);
      StringBuffer text = new StringBuffer();
      int i;
      while ((i = in.read()) != -1) {
        char c = (char)i;
        text.append(c);
      }
      in.close();
      fileIn.close();
      JTextArea ta = new JTextArea(text.toString());
      ta.setEditable(false);
      ta.setLineWrap(true);
      ta.setWrapStyleWord(true);
      return ta;
    }
    catch (IOException e) {
      return new JTextArea("Could not load help file");
    }
  }


  public void updateContents(Data data) {
    this.data = data;
    names.setListData(Utilities.convert(data.rules.keys()));
    rule.setText("");
  }

}
```

8.4.10 *gui/ReportBugAction.java*

```
/* Copyright (c) 1999 Jennifer E. Grucza
   All Rights Reserved. */

package DevelopmentConsultant.gui;

import DevelopmentConsultant.GlobalVars;
import DevelopmentConsultant.core.*;
import DevelopmentConsultant.rules.*;
import jess.*;
import com.ruleofeight.mail.*;
import javax.swing.*;
import javax.swing.border.*;
import java.awt.*;
import java.awt.event.*;
import java.util.*;
import java.io.*;

public class ReportBugAction extends AbstractAction implements Updateable {

  /** This class, along with ViewBugsInternalFrame, make up the
      bug reporting and tracking application of the system.  **/

  private Data data;
  private Component parent;

  private JComboBox name;
  private JTextField desc;
  private JComboBox platform;
  private JComboBox os;
  private JComboBox program;
  private JTextField version;
  private JComboBox module;
  private JComboBox urgency;
  private Hashtable symptoms;

  private JList choices;
  private JList chosen;

  public ReportBugAction(Data d, Component parentFrame) {
    super("Report a bug");
    data = d;
    parent = parentFrame;
  }

  public void actionPerformed(ActionEvent e) {
    /* first, setup window for report */

    JPanel panel = new JPanel(new GridLayout(8, 2));

    name = makeListEntry(panel, "Your name",
                ((Factory)data.factories.get("Users")).getInstances());
    desc = makeTextFieldEntry(panel, "One-line description of bug");
    platform = makeListEntry(panel, "Machine platform",
((Factory)data.factories.get("Platforms")).getInstances());
    os = makeListEntry(panel, "Operating System",
Systems")).getInstances();((Factory)data.factories.get("Operating
```

```
    program = makeListEntry(panel, "Program",
((Factory)data.factories.get("Programs")).getInstances());
    version = makeTextFieldEntry(panel, "Program version");
    module = makeListEntry(panel, "Affected program module",
                 ((Factory)data.factories.get("Modules")).getInstances());
    urgency = makeListEntry(panel, "Urgency",
((Factory)data.factories.get("Urgency
Descriptions")).getInstances());

    Box top = new Box(BoxLayout.Y_AXIS);
    top.add(panel);

    Factory f = (Factory)data.factories.get("Bug Symptoms");
    panel = new JPanel();
    symptoms = makeCheckBoxesEntry(panel, "Bug Symptoms", f.instances());
    top.add(panel);

    JPanel end = new JPanel();
    JTextArea detailedDesc = makeTextAreaEntry(end, "Detailed Description");
    top.add(end);

    Box all = new Box(BoxLayout.Y_AXIS);
    all.add(top);
    all.add(setupBottom());

    /* then user fills in information */
    int i = JOptionPane.showConfirmDialog(parent,
                                          all,
                                          "Report a bug",
                                          JOptionPane.OK_CANCEL_OPTION,
                                          JOptionPane.PLAIN_MESSAGE);

    /* and we send the report on its way */
    if (i == JOptionPane.OK_OPTION) {
      if (((DefaultListModel)chosen.getModel()).size() > 0) {
        try {
         EmailMessage mail = MailPack.newMessage(GlobalVars.outgoingMailServer);
          Instance inst = ((Factory)data.factories.get("Users")).
            getInstance((String)name.getSelectedItem());
          mail.setSender((String)inst.getPropertyValue("email address"));
          mail.setSubject(desc.getText());
          Enumeration enum = ((DefaultListModel)chosen.getModel()).elements();
          while (enum.hasMoreElements()) {
            inst = ((Factory)data.factories.get("Users")).
              getInstance((String)enum.nextElement());
            mail.addRecipient((String)inst.getPropertyValue("email address"));
          }

          StringBuffer content = new StringBuffer();
          content.append("The following bug report was created and sent ");
          content.append("using DevelopmentConsultant\n\n");
          addInfo(content, "Sender", (String)name.getSelectedItem());
          Date now = new Date();
          content.append("Date of report:  " + now.toString() + "\n");
          addInfo(content, "Urgency", (String)urgency.getSelectedItem());
          addInfo(content, "Platform", (String)platform.getSelectedItem());
          addInfo(content, "Operating System",
                  (String)platform.getSelectedItem());
          addInfo(content, "Program", (String)program.getSelectedItem());
```

```
        addInfo(content, "Version", version.getText());
        addInfo(content, "Affected module", (String)module.getSelectedItem());
        addCheckBoxInfo(content, "Bug Symptoms", symptoms);
        addInfo(content,
                "\nDetailed Description", "\n" + detailedDesc.getText());

        mail.setContent(content.toString());
        mail.send();
        data.bugReports.put(desc.getText(), content.toString());
      }
      catch (Exception ex) {
        GUIUtils.errorDialog(parent, ex);
      }
    }
  }
}

private void addInfo(StringBuffer s, String label, String content) {
  s.append(label).append(":  ").append(content).append("\n");
}

private void addCheckBoxInfo(StringBuffer s, String label, Hashtable ht) {
  s.append(label).append(":\n");
  Enumeration enum = ht.keys();
  while (enum.hasMoreElements()) {
    String key = (String)enum.nextElement();
    if (((JCheckBox)ht.get(key)).getModel().isSelected()) {
      s.append("\t " + key + "\n");
    }
  }
}

private JComboBox makeListEntry(JPanel panel, String label, Vector elements) {
  JComboBox cb = new JComboBox(elements);
  JLabel l = new JLabel(label, JLabel.LEFT);
  l.setLabelFor(cb);
  panel.add(l);
  panel.add(cb);
  return cb;
}

private JTextField makeTextFieldEntry(JPanel panel, String label) {
  JTextField tf = new JTextField(25);
  JLabel l = new JLabel(label, JLabel.LEFT);
  l.setLabelFor(tf);
  panel.add(l);
  panel.add(tf);
  return tf;
}

private JTextArea makeTextAreaEntry(JPanel panel, String label) {
  JTextArea ta = new JTextArea(15, 50);
  ta.setLineWrap(true);
  ta.setWrapStyleWord(true);
  JLabel l = new JLabel(label, JLabel.LEFT);
  l.setLabelFor(ta);
  l.setVerticalAlignment(JLabel.TOP);
  panel.add(l);
  panel.add(new JScrollPane(ta,
```

```
                                JScrollPane.VERTICAL_SCROLLBAR_ALWAYS,
                                JScrollPane.HORIZONTAL_SCROLLBAR_NEVER));
    return ta;
  }

  private Hashtable makeCheckBoxesEntry(JPanel panel,
                                        String label,
                                               Enumeration options) {
    Hashtable ht = new Hashtable();
    Box b = new Box(BoxLayout.Y_AXIS);
    JCheckBox check;
    String name;
    while (options.hasMoreElements()) {
      name = (String)options.nextElement();
      check = new JCheckBox(name);
      ht.put(name, check);
      b.add(check);
    }
    panel.add(new JLabel(label));
    panel.add(b);
    return ht;
  }

  private JPanel setupBottom() {
    JPanel p = new JPanel(new BorderLayout());
    chosen = new JList(new DefaultListModel());
    JScrollPane sp = new JScrollPane(chosen);
    sp.setPreferredSize(new Dimension(200, 200));
    JLabel l = new JLabel("Send To");
    l.setLabelFor(sp);
    Box left = new Box(BoxLayout.Y_AXIS);
    left.add(l);
    left.add(sp);
    Box upper = new Box(BoxLayout.X_AXIS);
    upper.add(left);

    Box middle = new Box(BoxLayout.Y_AXIS);

    JButton suggestB = new JButton("suggest recipients");
    suggestB.setAlignmentX(Component.CENTER_ALIGNMENT);
    suggestB.addActionListener(new ActionListener() {
      public void actionPerformed(ActionEvent e) {
        queryRBS();
      }
    });
    String tip = "Queries knowledge-base to find the users " +
      "most likely to be able to resolve this bug";
    suggestB.setToolTipText(tip);
    middle.add(suggestB);

    middle.add(Box.createVerticalStrut(20));

    JButton addB = new JButton("<= add");
    addB.setAlignmentX(Component.CENTER_ALIGNMENT);
    addB.addActionListener(new ActionListener() {
      public void actionPerformed(ActionEvent e) {
        DefaultListModel chosenList = (DefaultListModel)chosen.getModel();
        DefaultListModel choicesList = (DefaultListModel)choices.getModel();
        Object[] selected = choices.getSelectedValues();
```

```
      for (int i = 0; i < selected.length; i++) {
        chosenList.addElement(selected[i]);
        choicesList.removeElement(selected[i]);
      }
    }
  });
  middle.add(addB);

  JButton removeB = new JButton("remove =>");
  removeB.setAlignmentX(Component.CENTER_ALIGNMENT);
  removeB.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
      DefaultListModel chosenList = (DefaultListModel)chosen.getModel();
      DefaultListModel choicesList = (DefaultListModel)choices.getModel();
      Object[] selected = chosen.getSelectedValues();
      for (int i = 0; i < selected.length; i++) {
        choicesList.addElement(selected[i]);
        chosenList.removeElement(selected[i]);
      }
    }
  });
  middle.add(removeB);

  upper.add(middle);

  choices = new JList(Utilities.
                      vector2DLM(((Factory)data.factories.get("Users")).
                                 getInstances()));
  sp = new JScrollPane(choices);
  sp.setPreferredSize(new Dimension(200, 200));
  upper.add(sp);
  p.add(upper, BorderLayout.CENTER);

  EmptyBorder empty = new EmptyBorder(15, 15, 15, 15);
  p.setBorder(empty);
  return p;
}

private void queryRBS() {
  /* load all facts and rules into knowledge base from data */
  try {
    RBSEngine rbs = new RBSEngine();

    rbs.store("participants", new Vector());

    String rule = "(defrule find-participants\n" +
      "(definite-participant ?user)\n" +
      "=>\n" +
      "(call (fetch \"participants\") addElement ?user))";

    rbs.executeCommand(rule);
    Enumeration enum = data.rules.keys();
    while (enum.hasMoreElements()) {
      Object obj = enum.nextElement();
      rbs.executeCommand((String)data.rules.get(obj));
    }

    rbs.loadFacts(data);
    if (name.getSelectedItem() != null)
```

```
        rbs.assertString("(bug-sender-name \"" +
                         (String)name.getSelectedItem() + "\")");
      if (platform.getSelectedItem() != null)
        rbs.assertString("(bug-platform \"" +
                         (String)platform.getSelectedItem() + "\")");
      if (os.getSelectedItem() != null)
        rbs.assertString("(bug-os \"" +
                         (String)os.getSelectedItem() + "\")");
      if (program.getSelectedItem() != null)
        rbs.assertString("(bug-program \"" +
                         (String)program.getSelectedItem() + "\")");
      if (module.getSelectedItem() != null)
        rbs.assertString("(bug-module \"" +
                         (String)module.getSelectedItem() + "\")");
      if (urgency.getSelectedItem() != null)
        rbs.assertString("(bug-urgency \"" +
                         (String)urgency.getSelectedItem() + "\")");

      enum = symptoms.keys();
      while (enum.hasMoreElements()) {
        String key = (String)enum.nextElement();
        if (((JCheckBox)symptoms.get(key)).isSelected()) {
          rbs.assertString("(bug-symptom \"" + key + "\")");
        }
      }

      rbs.executeCommand("(watch rules)");
      rbs.executeCommand("(watch facts)");
      rbs.executeCommand("(run)");

      Value p = rbs.fetch("participants");
      Vector participants = (Vector)p.
        externalAddressValue(rbs.getGlobalContext());

      DefaultListModel chosenList = (DefaultListModel)chosen.getModel();
      DefaultListModel choicesList = (DefaultListModel)choices.getModel();

      enum = participants.elements();
      while (enum.hasMoreElements()) {
        String user = (String)enum.nextElement();
        if (choicesList.contains(user)) {
          chosenList.addElement(user);
          choicesList.removeElement(user);
        }
      }
    }
    catch (JessException je) {
      GUIUtils.errorDialog(parent, je);
    }
  }

  public void updateContents(Data data) {
    this.data = data;
  }
}
```

8.4.11 *gui/ViewBugsInternalFrame.java*

```
/* Copyright (c) 1999 Jennifer E. Grucza
   All Rights Reserved. */

package DevelopmentConsultant.gui;

import DevelopmentConsultant.core.*;
import javax.swing.*;
import javax.swing.event.*;
import java.awt.*;
import java.awt.event.*;
import java.util.*;

public class ViewBugsInternalFrame extends JInternalFrame implements Updateable

  /** This class handles display of all submitted bug reports.
      In the future it could be expanded with a search function, as
      well as being able to thread reports and replies.  **/

  private Data data;
  private JList indexes;
  private JTextArea report;
  private String selectedIndex;

  public ViewBugsInternalFrame(String name, Data d) {
    super(name, true, true, true, true);
    data = d;

    Vector v = Utilities.convert(data.bugReports.keys());
    Collections.sort(v);
    indexes = new JList(v);
    indexes.setSelectionMode(ListSelectionModel.SINGLE_SELECTION);
    indexes.addListSelectionListener(new ListSelectionListener() {
      public void valueChanged(ListSelectionEvent e) {
        selectedIndex = (String)indexes.getSelectedValue();
        if (selectedIndex != null) {
          report.setText((String)data.bugReports.get(selectedIndex));
        }
        else {
          report.setText("");
        }
      }
    });
    JScrollPane s1 = new JScrollPane(indexes);
    s1.setPreferredSize(new Dimension(200, 600));

    report = new JTextArea();
    report.setLineWrap(true);
    report.setWrapStyleWord(true);
    JScrollPane s2 = new JScrollPane(report,
                                     JScrollPane.VERTICAL_SCROLLBAR_ALWAYS,
                                     JScrollPane.HORIZONTAL_SCROLLBAR_NEVER);

    JSplitPane sp = new JSplitPane(JSplitPane.HORIZONTAL_SPLIT, s1, s2);
    getContentPane().add(sp, BorderLayout.CENTER);
  }

  public void updateContents(Data data) {
    this.data = data;
```

```
    Vector v = Utilities.convert(data.bugReports.keys());
    Collections.sort(v);
    indexes.setListData(v);
  }
}
```