ELSEVIER

Expert Systems with Applications xx (0000) xxx–xxx

Expert Systems with Applications

www.elsevier.com/locate/eswa

A GUI for Jess

Jelena Jovanović, Dragan Gašević, Vladan Devedžić*

FON, School of Business Administration, University of Belgrade, P.O. Box 52, Jove Ilića 154, 11000 Belgrade, Yugoslavia

Received 2 October 2003; revised 2 October 2003; accepted 1 December 2003

Abstract

The paper describes JessGUI, a graphical user interface developed on top of the Jess expert system shell. The central idea of the JessGUI project was to make building, revising, updating, and testing Jess-based expert systems easier, more flexible, and more user friendly. There are many other expert system building tools providing a rich and comfortable integrated development environment to expert system builders. However, they are all either commercial or proprietary products. Jess and JessGUI are open-source freeware, and yet they are well suited for building even complex expert system applications, both stand-alone and Web-based ones. An important feature of JessGUI is its capability of saving knowledge bases in XML format (in addition to the original Jess format), thus making them potentially easy to interoperate with other knowledge bases on the Internet. Jess and JessGUI are also used as practical knowledge engineering tools to support both introductory and advanced university courses on expert systems. The paper presents design details of JessGUI, explains its links with the underlying Jess knowledge representation and reasoning tools, and shows examples of using JessGUI in expert system development. It also discusses some of the current efforts in extending Jess/JessGUI in order to provide intelligent features originally not supported in Jess.

Keywords: Expert systems; Development tools; Graphical user interface; Knowledge base interoperability

1. Introduction

There is an entire hierarchy of expert system (ES) development tools in terms of the level of flexibility they provide in building ES and the range of knowledge representation and reasoning techniques they support. Simple ones include special-purpose programming languages for ES development that facilitate knowledge representation and reasoning, as well as extensions of general-purpose programming languages (such as C and C++) in order to provide language constructs and library functions, classes and methods to support building ES. For example, Hamada et al. (1995) have developed a number of C++ classes and methods to support representing rules, working-memory elements, and other knowledge elements, as well as reasoning techniques. These facilitate ES development and enable 'direct coding', i.e. inserting C++ statements into production rules. Another example is Rete++, a forward and backward chaining inference engine based on the famous Rete pattern-matching algorithm for production systems (Forgy, 1982), developed as a fully encapsulated extension to C++ (Haley Enterprise, 1996). Cafe Rete, from the same manufacturer, is a Java class library that seamlessly integrates a rules engine within Java applications, servlets, EJBs, etc.

In the middle part of the hierarchy are specific AI programming languages and tools, such as Prolog, LOOM, and Parka. Prolog interpreter has a built in backward chaining inference engine that processes Prolog rules and enables automatic backtracking, hence ES can be developed in Prolog. LOOM is a language and an environment for constructing ES and other intelligent applications, with built-in techniques for representing knowledge as definitions, rules, and facts, as well as with a built-in Prolog-technology deductive reasoning engine (Yen, Juang, & MacGregor, 1991). Parka and Parka-DB are frame-based AI languages/tools that enable scaling knowledge bases up to extremely large-size applications, and use DBMS technologies to support inferencing and data management (Hendler, Stoffel, Taylor, Rager, & Kettler, 1997).

The upper parts of the hierarchy are occupied by integrated, rich ES development environments, supporting and combining several ES paradigms, as well as different mechanisms for representing and handling uncertainty,

* Corresponding author. Tel.: +381-11-3950853; fax: +381-11-461221.
E-mail addresses: devedzic@galeb.etf.bg.ac.yu (V. Devedžić); jeljov@yubc.net (J. Jovanović); gasevic@yahoo.com (D. Gašević).

0957-4174/$ - see front matter © 2003 Published by Elsevier Ltd.
doi:10.1016/j.eswa.2003.12.012

providing explanations, and enabling automatic knowledge-base construction and updating by means of machine learning. Examples of such tools are commercial products, such as Exsys CORVID (http://www.exsys.com) and Vanguard Software DecisionPro environment (http://www.vanguardsw.com/). For more information about ES building tools at different levels of sophistication and integration, see general ES literature (Durkin & Durkin, 1998; Giarratano, 1998), as well as Internet resources listed at AAAI site (http://www.aaai.org/aitopics/html/expert.html) and at PC AI site (http://www.pcai.com/web/ai_info/expert_systems.html).

In recent years, *Java Expert System Shell*, or *Jess* (Friedman-Hill, 2002; Sandia, 2003) has become a popular development tool for ES. Jess is essentially a reimplementation of a subset of the earlier CLIPS shell (CLIPS, 2003) in Java. Its reasoning is based on a list of known facts and a set of rules that try to match on these facts in its fact base. Rule-based reasoning of Jess inference engine is mostly Rete-based forward chaining, but backward chaining is supported as well.

Jess is a simple, yet powerful enough tool to allow for building a number of industry-strength ES applications (Friedman-Hill, 2002). Its major advantage is its capability to easily integrate with other Java programs through its well-defined API for controlling the reasoning engine from Java (Eriksson, 2003). Java programs can send expressions to the Jess inference engine for evaluation, and it is easy to extend Jess with new functions in Java because it is an open-source freeware. In addition, Jess implements some additional functionality not provided by CLIPS.

However, Jess lacks a GUI. A couple of members of the GOOD OLD AI research group (http://goodoldai.org.yu), based at the University of Belgrade, Serbia and Montenegro, have ventured into an R&D project of designing and developing a GUI for Jess, called *JessGUI*, suitable for all platforms that support Java Virtual Machine. JessGUI v. 1.0 is now complete, and is already used in practical projects and as a teaching tool. This paper describes design and implementation of JessGUI v. 1.0 and experiences so far with using it along with Jess in practice.

The paper is organized as follows. Section 2 defines precisely what we wanted JessGUI to provide, enable, and support. Section 3 briefly overviews some other current efforts related to ES and ES shells GUI, and more specifically to Jess and its applications. Sections 4 and 5 describe the overall organization of JessGUI, its communication with Jess' built-in knowledge representation techniques and reasoning mechanisms, and details of its design. Section 6 covers one of JessGUI's most important features, using eXtensible Markup Language (XML) advantages to ensure for easy interoperability between Jess-based systems and other XML applications. Section 7 discusses experiences with using Jess/JessGUI so far and advantages and disadvantages noticed. Section 8 summarizes the paper and indicates directions for future development of JessGUI.

2. Problem statement

JessGUI project objectives include the following:

- the GUI should make Jess knowledge base building easy for developers, with minimum requirements in terms of knowledge of details of Jess knowledge representation format;
- from the ES developers' perspective, the GUI should look as an integral part of the shell (i.e. it is the Jess/JessGUI *combination* that the ES builders normally use as the development environment);
- it should be easy to integrate with Jess' built-in knowledge representation and reasoning tools;
- it should enable building and running both stand-alone and Web-based ES applications;
- it should undergo a thorough testing through development of a number of simple practical applications;
- it should facilitate interoperability between Jess knowledge bases and external Web applications;
- it should be open for further development, extensions, and integration with external intelligent tools.

JessGUI is now being constantly tested, evaluated, and maintained based on the users' comments and suggestions. The project is developing in a wider context of related R&D efforts both within the GOOD OLD AI group and elsewhere.

3. Related work

3.1. Web-based expert systems

One important line of ES research that we follow in developing JessGUI is related to Web-based ES. To deploy a Web-enabled ES, there are a number of architectural approaches from which we may want to start. The most common is the HTML-CGI architecture: The user interacts with HTML entry forms in a Web browser; information entered by the user is sent to the Web server which forwards it to the CGI (Common Gateway Interface) program which then replies with new HTML pages (Alpert, Singley, & Fairweather, 2000; Eriksson, 1996). All the expert functionality resides on the server side (in the CGI program), but the user interacts with it using a standard Web browser.

Another option might be distributed client–server architecture—a downloadable Java applet contains the user interaction portion of the Expert system, and communicates directly with the server application using a socket connection or other inter-program communication mechanism—some of the expert behavior resides in the client, some in the server (Eriksson, 1996; Potter et al., 2001).

3.2. Knowledge interchange and interoperability

The second line of research that we follow closely is related to the efforts of making the knowledge represented

on the Web interoperable and ready to be shared between applications. One such effort is *The Rule Markup Initiative* (http://www.dfki.uni-kl.de/ruleml/) whose mission is to define a shared *Rule Markup Language* (*RuleML*), permitting both forward (bottom-up) and backward (top-down) rules in XML for deduction, rewriting, and further inferential-transformational tasks (Boley, 2001). Another important one is *The eXtensible Rule Markup Language*, or *XRML*, that enables identification of implicit rules embedded in Web pages, interchanging them with structured-format rule-based systems, and accessing them by different applications (Lee & Sohn, 2003).

3.3. Extensions, upgrades, and applicability of Jess

We also monitor recent efforts related to integrating Jess with other development environments, as well as contributions of other research groups to the evolution of Jess itself. The most notable recent work in that direction is presented by Eriksson (2003). He has developed a plug-in called *JessTab* (www.ida.liu.se/~her/JessTab), which integrates Jess with Protégé-2000, a popular, modular ontology development and knowledge acquisition tool developed at Stanford university (http://protege.stanford.edu/). JessTab enables a Jess engine to run inside the Protégé-2000 framework and lets users build knowledge bases in Protégé-2000 that work with Jess programs and rule bases. It takes advantage of the Jess API to map information in the Protégé knowledge base to Jess facts and to extend Jess with additional functions for communication with Protégé-2000 (Eriksson, 2003).

Program *JessWin*[1] represents probably the first attempt to introduce certain graphical elements in the Jess development environment, thus making it more user-friendly. However, its graphics, reduced to the use of windows, is fairly modest and as such does not significantly enhance the quality of interaction between the user and Jess. *JessWin* users still need perfect knowledge of Jess syntax in order to create valid ES. Another important contribution to Jess evolution is *FuzzyJess*, an extension of Jess that enables usage of fuzzy facts. It was developed through integration of Jess and *NRC FuzzyJ Toolkit*,[2] a set of Java(tm) classes that provide the capability for handling fuzzy concepts and reasoning. It is also worth noting that Jess' capabilities were further extended with *JavaMailFunctions*,[3] new user defined functions interfacing to Sun's JavaMail 1.1 API.

In order to integrate two Java-supported technologies, Jess and Java XML parser, Leff (2001) developed Jess User Functions that load XML documents and convert the Document Object Model (DOM) tree into a series of Jess facts. A fact is created for each XML tag and each attribute found in the document loaded. Then ordinary Jess rules can be used to reason about the XML document loaded. Also rules themselves can be expressed in XML. Detailed examples using these XML/Jess extensions can be reached at http://ecitizen.mit.edu/ecap3.html. Left's approach is useful because it demonstrates an integration of XML and Jess, but it cannot be used for representing all Jess' knowledge base features (e.g. relations between a template and its instances, function definitions, etc.). The importance of integrating Jess with XML is also discussed in the experiences of other developers at http://herzberg.ca.sandia.gov/jess/devlog.shtml. In this context, XML support means to be able to convert easily from Jess scripts to an XML representation and back. To implement this, again, what is needed is a fast, flexible parser, with excellent error reporting and a public API.

3.4. Research context of JessGUI development

The idea of developing JessGUI emerged along with other important research activities and results achieved by the GOOD OLD AI group—many of the group's activities are closely related to ES technology. Devedžić and Radović (1999) have proposed a multi-layered framework for building intelligent systems, called *OBOA*, which incorporates a number of ES techniques. More recently, a number of fuzzy logic tools have been developed in accordance with the OBOA framework; they make the basis of the more specific *Fuzzy OBOA* framework (Šendelj, 2003). A fuzzy ES development tool called *FES* is developed to fit Fuzzy OBOA and used to implement a couple of fuzzy ES in a medical domain. Jess has been explicitly used in *Code Tutor*, a Web-based intelligent tutoring system for fast students' briefing in the area of radio-communication (Šimić & Devedžić, 2003). A novel design of forward-chaining rule-based inference engine has been implemented as an interoperable software component (Čakić & Devedžić, 1999), and a Web-based ES for diagnosis of car malfunctioning was developed (Andrić, Devedžić, & Andrejić, 2003)).

4. Proposed solution

The main idea of the JessGUI project was to develop a GUI for Jess ES Shell which would make this ES development environment more user friendly and much easier to work with, hence enlarging the number of its potential users.

Important advantages of the new user interface (UI) in comparison with the existing one are as follows:

- Interaction and dialogs using familiar graphical elements instead of plain command prompt—JessGUI enables

[1] Developed by William E. Wheeler, it can be freely downloaded from the Jess official web site http://herzberg.ca.sandia.gov/jess under the link 'Users' contributions'.

[2] The toolkit was developed at the National Research Council of Canada's Institute for Information Technology.

[3] Written by Thomas Barnekow, they can be downloaded from http://herzberg.ca.sandia.gov/jess/user.shtml.

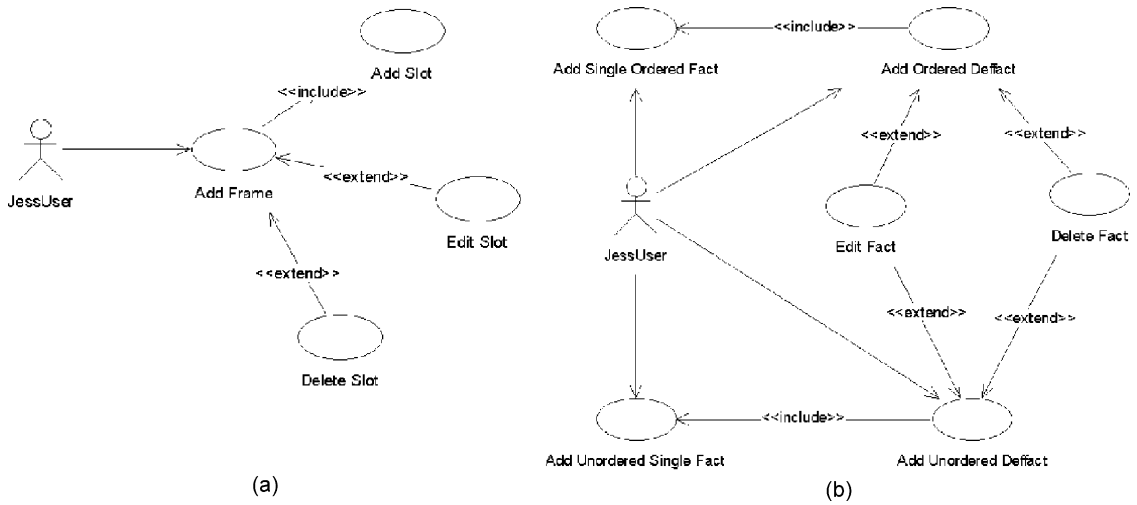

Fig. 1. Use-case diagrams representing complex UCs: (a) 'Add Frame' (b) 'Add Facts'.

users to work with windows, buttons, text fields and other graphical objects which are essential parts of UI of modern applications. It is much more convenient and less time-consuming than typing complicated constructs at the command prompt.

- Users need only the knowledge of the basic Jess concepts instead of the complete syntax of the CLIPS language: JessGUI offers an alternative to learning complex structures that are part of CLIPS, which Jess has inherited from its predecessor.

JessGUI design was based on the layout of Protégé-2000s GUI. Just like the UI of Protégé-2000, JessGUI has a few panels, one for each of Jess' concepts.

4.1. Use cases

The main use cases (UC) of the JessGUI application, implementing the options of the main menu, are:

- Create new ES Project
- Open existing ES Project
- Save ES Project
- Run (Start Jess Engine)

UC 'Create New ES Project' is a complex UC and can be further decomposed into separate UCs each representing the creation of one of the Jess' concepts. This decomposition reveals the following UCs:

- Add Module
- Add Global Variable
- Add User Function
- Add Rule
- Add Frame
- Add Facts

The first four UCs are rather simple (elementary) and not particularly interesting. The last two UCs need further clarification.

The 'Add Frame' UC (Fig. 1a) describes the creation of the Jess concept called *frame*,[4] that represents a template for instantiating *unordered facts* (see Section 4.2 for more details). Each frame is composed of one or more *slots*.[5] This UC includes the elementary UC 'Add Slot', and can be extended by another two elementary UCs, 'Edit Slot' and 'Delete Slot'.

'Add Facts' is the most complex UC. It is further decomposed into separate UCs that describe the process of instantiating ordered and unordered facts, both as single facts or as groups of facts (i.e. through deffacts structure). Its decomposition is presented in the use case diagram in Fig. 1b.

4.2. Links between JessGUI and jess

JessGUI does not directly operate with Java API that Jess provides for representing its main concepts (**Section 5.3). Instead of using Jess' classes for representing rules, frames and other concepts, JessGUI introduces its own classes for the purpose of representing these concepts.

Since the Jess inference engine can process a knowledge base only if it is represented either with classes that Jess provides for that purpose, or as a document in CLIPS format (.clp file), it was essential to transform data stored in instances of JessGUI classes to one of the formats just mentioned. As an important early design decision was to

[4] The original name of this concept in Jess terminology is 'template', but in the JessGUI project the term 'frame' is used in order to achieve higher-level generalization and conformance with a wider spectrum of AI community developers.

[5] The concept of frames and slots is similar to the concept of records and their fields in standard programming languages.

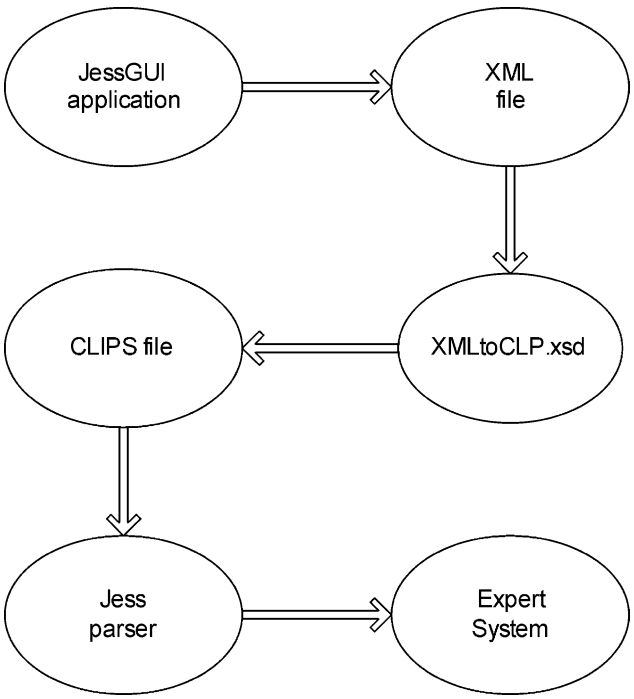

Fig. 2. The major link between JessGUI and Jess.

enable permanent storage of knowledge bases in XML documents for the purpose of their easier reusability, the painless solution of the above dilemma was to take advantage of eXtensible Stylesheet Language Transformations (XSLT, http://www.w3.org/TR/xslt) and transform XML files into the corresponding CLIPS files familiar to the Jess' inference engine. Fig. 2 depicts this idea.

Jess enables building rule-based ES whose knowledge base is composed of two important structures: facts and rules. A *fact* is a construct that defines a piece of information that is known to be true, whereas a *rule* is an *if/then* statement that defines the set of facts that must be true (the *if* part) before a set of actions (the *then* part) can be executed (Friedman-Hill, 2003). In Jess, there are three kinds of facts: *ordered facts*, i.e. facts without a predefined structure; *unordered facts*, i.e. facts whose construction is based on using frames or templates; *definstance facts*, i.e. facts that are actually instances of user-defined Java classes. JessGUI v.1.0 supports creating the first two types of facts.

From JessGUI, one can run both forward- and backward-chaining inferences that Jess' inference engine provides. This is configurable—before choosing the 'Run' menu option, the user should set the desired configuration of the inference engine in the 'Options' menu. Since JessGUI was conceptualized as an extensible ES building tool, providing support for Jess' native inference methods presents just the first step in its development. Application is open for further extensions and its capabilities will be enhanced by adding support for other inference methods, some of them developed by members of the GOOD-OLD-AI group.

5. Design details

Classes created during the development of JessGUI can be categorized into two main categories:

1. presentation layer classes, i.e. classes that implement frames and dialogs of the JessGUI application;
2. middle layer classes, i.e. classes that implement the application logic.

The classes corresponding to the third layer of the classical concept of three-tier software architecture, i.e. the classes that should support communication with persistent data storage, were not implemented. Instead of storing data in a database, JessGUI enables its users to store their projects in the form of.xml or.clp files.

5.1. Presentation layer

Presentation layer classes are shown in Fig. 3. For the sake of clarity, the diagram shows only the classes that implement the most important panels and dialogs of the developed graphical user interface. Analyzing the diagram, one can notice that each Jess concept has its associated panel or dialog and a class that implements that panel/dialog.

5.2. Application layer

Application-layer classes can be further divided into the following two categories:

1. classes that represent Jess' main concepts;
2. controllers and mediators.

5.2.1. Representing Jess concepts

JessGUI application does not use classes that Jess ES Shell provides for representing its main concepts (rules, facts, templates, etc.). Instead, at the level of the application logic, there are classes that represent each of those concepts (one class per concept). These classes, through their attributes and methods, support storing all data and knowledge that the knowledge-base developer provides. Decision not to work with Jess' classes was based on the following reasons:

- classes that Jess provides for representing some of its concepts cannot be directly instantiated—only Jess parser can create their instances while parsing the file that contains ES code (file with.clp extension);
- achieving higher level of generalization and reusability—the knowledge base created using JessGUI can be processed by different inference engines other then the one that Jess uses.

The class diagram shown in Fig. 4 depicts the structure and interconnections of previously discussed JessGUI classes that implement the basic Jess concepts. In order to

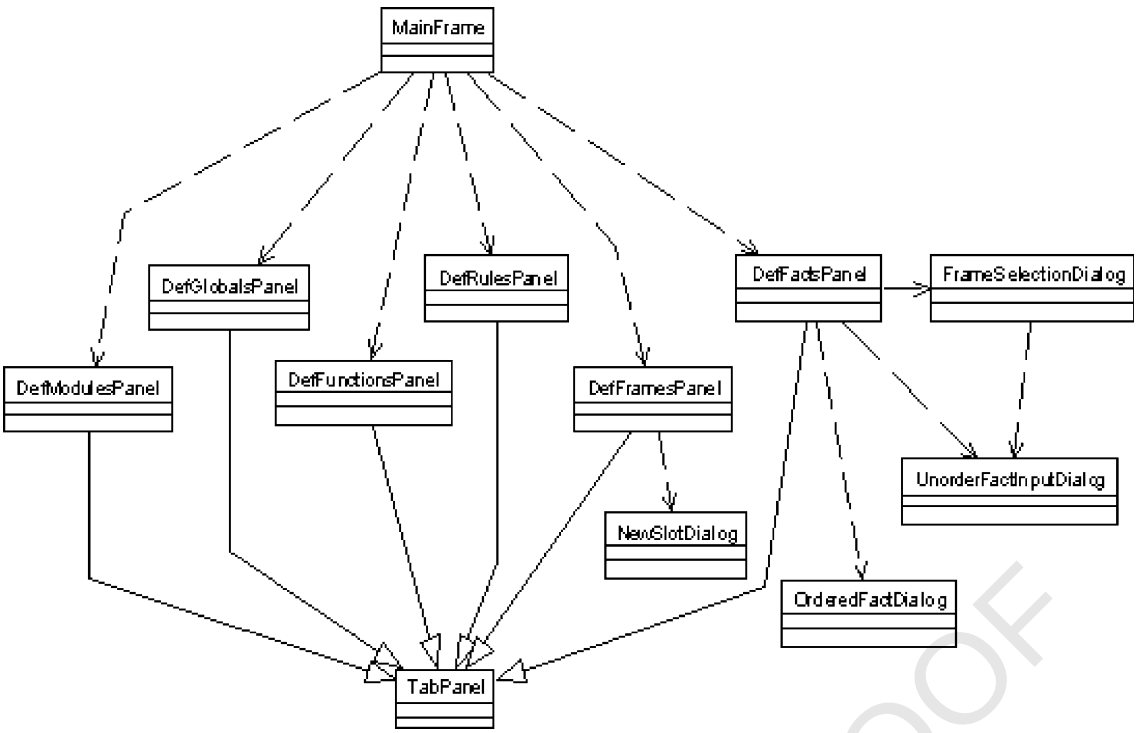

Fig. 3. Presentation layer classes.

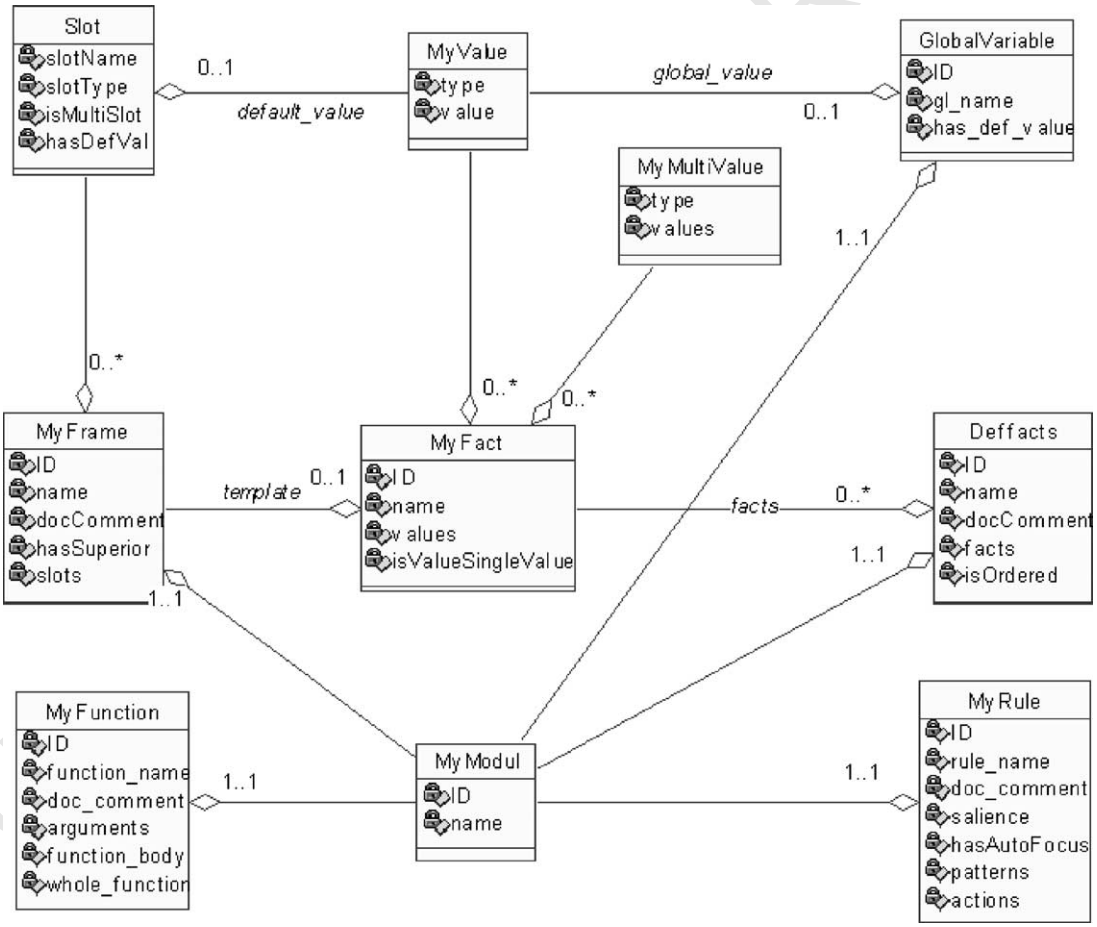

Fig. 4. Classes that represent the main Jess concepts.

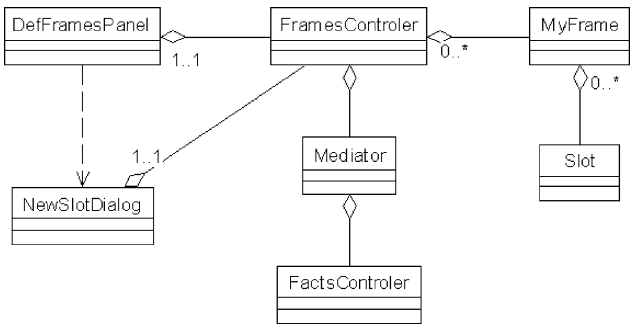

Fig. 5. Classes that implement the 'Add Frame' use case.

avoid clutter, the class diagram does not show the methods for setting and getting attribute values. A quick study of attribute lists of the classes shown on the diagram is enough to confirm that these classes can capture all data necessary for constructing the concepts they represent.

The class diagram shown in Fig. 5 represents the classes that cooperate in implementing the complex use case 'Add Frame'. The classes 'FrameControler' and 'Mediator' are responsible for providing low-level coupling between presentation and application logic layers.

5.2.2. Controllers and mediators

A software pattern is named problem/solution pair that can be applied in a new problem situation (Larman, 1997). Patterns are general principles and solutions that are used in software development.

General Responsibility Assignment Software Patterns, or GRASP patterns, aim to resolve problems related to assigning responsibilities to objects during software design. All main GRASP patterns, namely 'Expert', 'High Cohesion', 'Low Coupling' and 'Controller', were used while designing JessGUI. For example, JessGUI uses one controller class for governing each implemented use case, as well as one class that represents global controller that coordinates work of other controller classes and manages main operations.

Design of JessGUI was based also on well-known design patterns, or GoF patterns. They represent descriptions of communicating objects and classes that are customized to solve a general design problem in a particular context (Gamma, Helm, Johnson, & Vlissides, 1995). For example, JessGUI takes advantages of the 'Mediator' pattern that introduces an object responsible for interaction between a set of objects, thus promoting low-level coupling between them. JessGUI's classes 'Mediator', 'FramesControler' and 'FactsControler' implement this pattern. Their interconnection can be observed in Fig. 5.

5.3. Software organization and packages

Software classes developed as a part of JessGUI project are logically organized into three main groups:

1. classes that implement panels and dialogs of the user interface;
2. classes responsible for handling system events and supporting work with XML files;
3. classes that represent the main concepts of the Jess ES Shell.

This logical decomposition can be graphically represented using software packages. The package with classes from the first group mentioned above is shown in Fig. 6.

5.4. Example screens of JessGUI

In order to illustrate interface design and look-and-feel of JessGUI, Figs. 7a and 8 show panels for creating frames and rules, respectively.

Fig. 7a shows a screenshot of JessGUI, while the panel 'Frames' is active. This panel enables users to define the structure of frames that would later serve as templates for creating unordered facts. The user has to enter a name for the frame and optionally a description that should clarify the purpose of the frame. Each frame contains one or more slots. The user creates them by pressing the 'Add' button, thus invoking a special-purpose dialog, shown in Fig. 7b, for defining features for each slot. Each slot, after its features have been specified, is represented in the table that occupies the central part of the panel. The user has also an option to derive a new frame from an existing one. In that case, the new frame inherits all slots from its 'parent' frame and

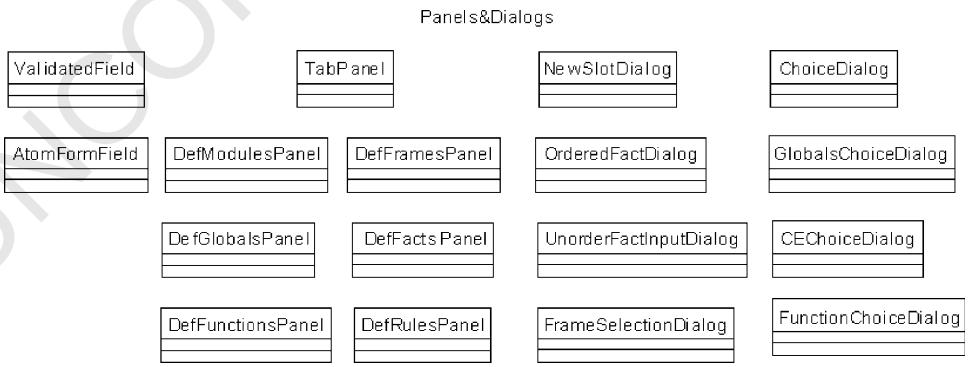

Fig. 6. The package with presentation layer classes.

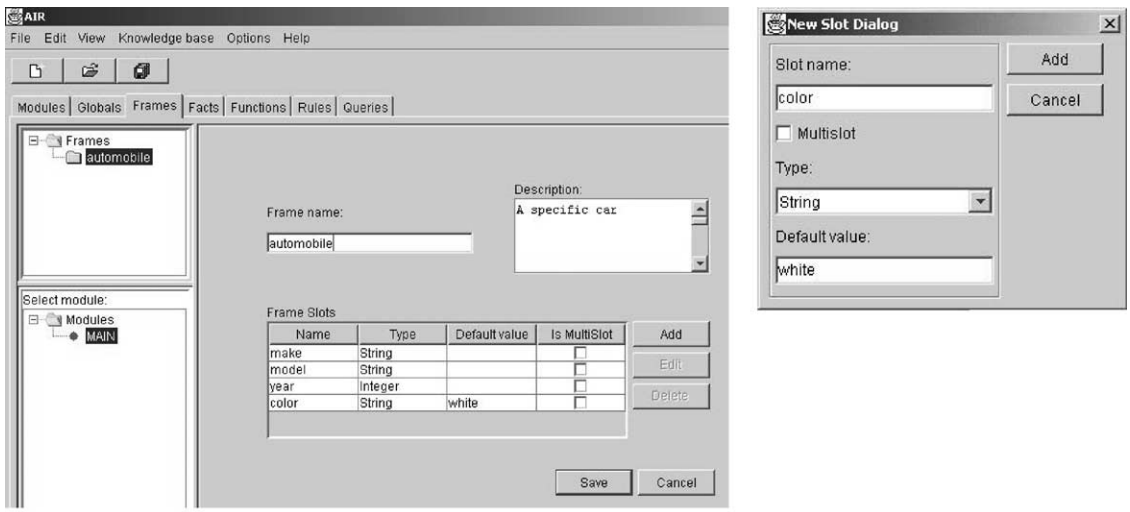

Fig. 7. (a) Panel for specifying frame's properties. (b) Dialog for slot definition.

automatically adds these inherited features to the frame, so the user should specify only the new slots.

Fig. 8 shows the panel 'Rules' for creating rules as the principal knowledge elements of the Jess knowledge base. As it can be noticed in the figure, this panel enables users to specify the rule's name, priority, whether it will have auto-focus or not and optionally to provide its short description. After making these specifications, the user should define conditions (IF-part) and actions (THEN-part) of the rule. After creating a rule, the user can check its validity before saving it, by pressing the 'Validate' button that would be a signal for the system to initiate parsing the rule and producing a message, in the form of dialog box, to inform the user of the results of the parsing process.

6. Interoperability issues

An important aspect of JessGUI is interoperability, i.e. enabling other Web (or non-Web) applications to access and use Jess knowledge base. Jess knowledge base is normally represented in Jess/CLIPS format. However, for parsing

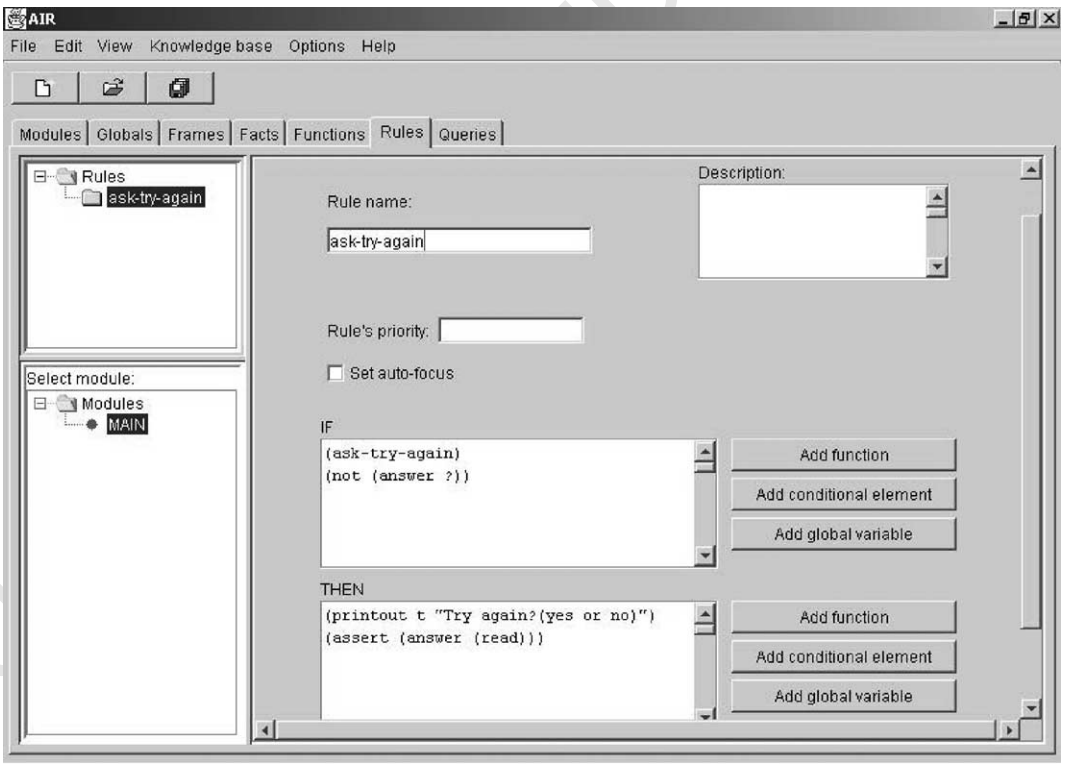

Fig. 8. Panel that supports rule creation.

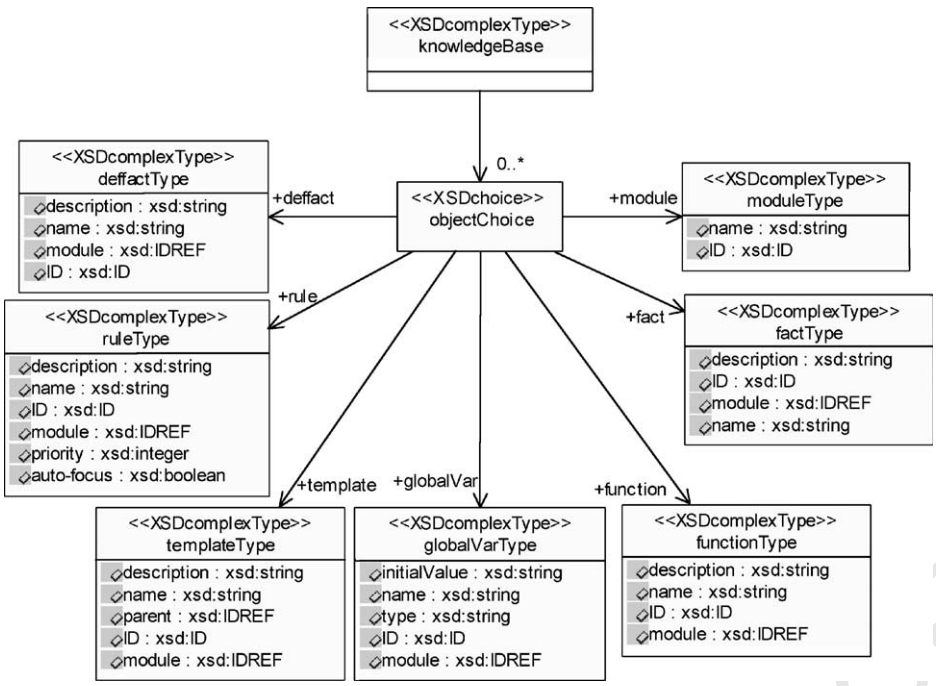

Fig. 9. UML class diagram of the root element in the JessGUI XML format.

such documents one must develop his/her own parser, which is difficult and time-consuming. It is more efficient to use XML-based knowledge-base representation, because there are many freeware XML parsers that can be used in a number of applications. Furthermore, XML documents (i.e. knowledge base) can be transformed to other XML or non-XML documents using XSLT. Of course, XML is also a fundamental technology for knowledge representation on the Semantic Web.

We developed an XML format for Jess knowledge bases using W3C's XML Schema for defining XML document grammar, and made it available to Jess knowledge-base builders through JessGUI. This XML format can contain and represent all Jess concepts (e.g. module, global variable, etc.). Speaking in terms of XML Schema definition, for each of these concepts we defined a proper complex type. We used UML profile for modeling XML Schema (Carlson, 2001) in order to obtain a better readability as well as a better documentation of the JessGUI's XML format. Fig. 9 depicts a class diagram for the content model of the XML format root element (i.e. *knowledgeBase*). This element can contain an unbounded number of XML elements: *module*, *globalVar*, *deffact*, *fact*, *template*, *function*, and *rule*. Each of these elements represents a real Jess concept. In Fig. 9 we show only a part of the XML Schema definition, while other elements are also defined using a similar procedure.

Each JessGUI's class is responsible for saving the state of its objects' attributes in XML format through its dedicated a *save* method. Fig. 10 shows how this is implemented in the *Rule* class. One can note that we do not dynamically create a DOM tree, but we save XML documents as a regular text file. This way we prevent potential problems in the case we have a large knowledge base (i.e. a large DOM tree). After executing the *save* method from Fig. 10 for a real-application JessGUI rule, an XML document is produced that looks as in Fig. 11.

Having XML-based format for JessGUI is a good feature for interoperability with other knowledge-based systems, but this format cannot be used by the Jess interpreter. The Jess interpreter in its basic distribution interprets only Jess/CLIPS code. Thus, we need a way to transform the JessGUI XML format into the Jess/CLIPS code. XSLT is a natural solution for this problem. We developed an XSLT that transforms JessGUI XML format into Jess/CLIPS format, and this XSLT is performed in JessGUI when one wants to export a JessGUI knowledge base into Jess/CLIPS format. A part of this XSLT (i.e. XSLT template) that transforms JessGUI XML *deffact* into Jess/CLIPS *deffact* statement looks as in Fig. 12.

Developing similar XSLTs we can support interoperability with other Semantic Web tools. For example, if one wants to import a Jess/JessGUI knowledge base into Protégé-2000, then it is enough to develop an XSLT from JessGUI format to RDF(S) format that can be imported in Protégé-2000. Of course, in this example there is a constraint that Protégé-2000 can import only templates and unordered facts because RDF(S) is being used. Importing a Jess knowledge base in Protégé-2000 can be useful, because it is a way to produce an ontology (Devedžić, 2002). Other limitations can occur in the case of a knowledge base interchange with some other systems. This is because other systems do not support all Jess features. Currently, we make efforts towards achieving interoperability between JessGUI and JavaDON (a tool for

```
public class Rule
{
  private String ID;
  private String rule_name;
  private String doc_comment;
  private int salience;
  private boolean hasAutoFocus;
  private String patterns;
  private String actions;
  private Modul modul;

  /*...*/
  public void save(Writer out) throws IOException
  {
    out.write("<rule name=\"" + rule_name + "\" ID=\"" + ID + "\"
    module=\"" + modul.getID() + "\" priority=\"" + salience + "\"
    auto-focus=\"" + hasAutoFocus + "\">\n");
    out.write("<description>" + doc_comment + "</description>\n");
    out.write("<definition>\n");
    out.write("<condition>\n");
    out.write("<value>" + patterns + "</value>\n");
    out.write("</condition>\n");
    out.write("<action>\n");
    out.write("<value>" + actions + "</value>\n");
    out.write("</action>\n");
    out.write("</definition>\n");
    out.write("</rule>\n");
  }
  /*...*/
}
```

Fig. 10. An example of JessGUI class and its method for saving in XML format.

developing intelligent systems developed by GOOD OLD AI members and based on OBOA framework). Since JavaDON also uses XML-based format for knowledge base it is enough to develop an appropriate XSLT.

7. Practical experience with JessGUI

So far, we used JessGUI in practical developments in two ways. First, in order to improve and stabilize JessGUI itself, we re-developed the knowledge bases of some ES we developed earlier, this time using Jess/JessGUI. An example is Defector, an ES for diagnosing malfunctions on cars, http://galeb.etf.bg.ac.yu/~aandreja/defektator/defektator.htm (Andrić et al., 2003). Fig. 13 shows a screenshot from Defector. Such re-developments were comparatively easy since we started from working systems, but were also very important for us in terms of noticing and correcting some weaknesses of JessGUI from the developers' perspective.

Second, we used Jess/JessGUI for developing new ES. Two of them are a partner-matching ES (currently under development and testing) and Travel Guide, a just completed ES that recommends tourists destinations, means of transportation, facilities, activities, and the like

```
<?xml version = "1.0" encoding = "UTF-8"?>
<knowledgeBase>
      <module ID="J1" name="MAIN" />
      <!-- ... -->
      <rule name="player-select" ID="J7" priority="1"  module = "J1"
      auto-focus="true">
            <code>
            (defrule player-select
                  (temp)
                  =>
                  (printout t "Who moves first (Computer: c "
                  "Human: h)? ")
                  (assert (player-select (read)))
            )
            </code>
      </rule>
      <!-- ... -->
</knowledgeBase>
```

Fig. 11. An example an XML document is produced by saving a JessGUI rule in XML format.

```
<xsl:template match="deffact">
      <xsl:param name="ID">null</xsl:param>
      <xsl:if test="@module=$ID">
            (deffact <xsl:value-of select="@name"/>
            "<xsl:value-of select="description"/>"
            <xsl:for-each select="factRef">
                  <xsl:variable name="IDtemp">
                        <xsl:value-of select="./@ID"/>
                  </xsl:variable>
                  (<xsl:value-of select="/knowledgeBase/fact[@ID =
                  $IDtemp]/@name"/>)
            </xsl:for-each>
            )
      </xsl:if>
</xsl:template>
```

Fig. 12. An example XSLT template of JessGUI's format converter.

(Fig. 14). Both systems are large-scale systems and involved a number of developers. Their evaluations of JessGUI were an invaluable feedback on JessGUI's usability.

Jess/JessGUI is already in use in teaching ES courses at three higher-education institutions. The students are exercise simple ES development and understanding of basic knowledge representation and reasoning techniques using Jess/JessGUI in the labs, and also use it to do their required projects. We have already conducted informal evaluations of Jess/JessGUI with students. They were asked to compare Jess' native command-line UI to JessGUI. Along with positive feelings about JessGUI and stressing more comfortable work with that without JessGUI, the students have also provided critical opinions that were extremely useful for improving JessGUI. Being a still evolving ES building tool, JessGUI naturally still has drawbacks that are either in the process of elimination or are planed to be resolved in the nearest future.

The current version of the JessGUI program does provide a certain level of help in order to simplify the process of creating the knowledge base, but not full support. The support that JessGUI provides are three dialogs that enable users to browse through the list of Jess-intrinsic and user-defined functions, Jess conditional elements and previously defined global variables (if any), respectively. This is probably the main drawback of JessGUI, since it imposes the burden of learning CLIPS syntax on its users. The problem stems from the fact that the members of the GOOD OLD AI group, not having at their disposal the complete

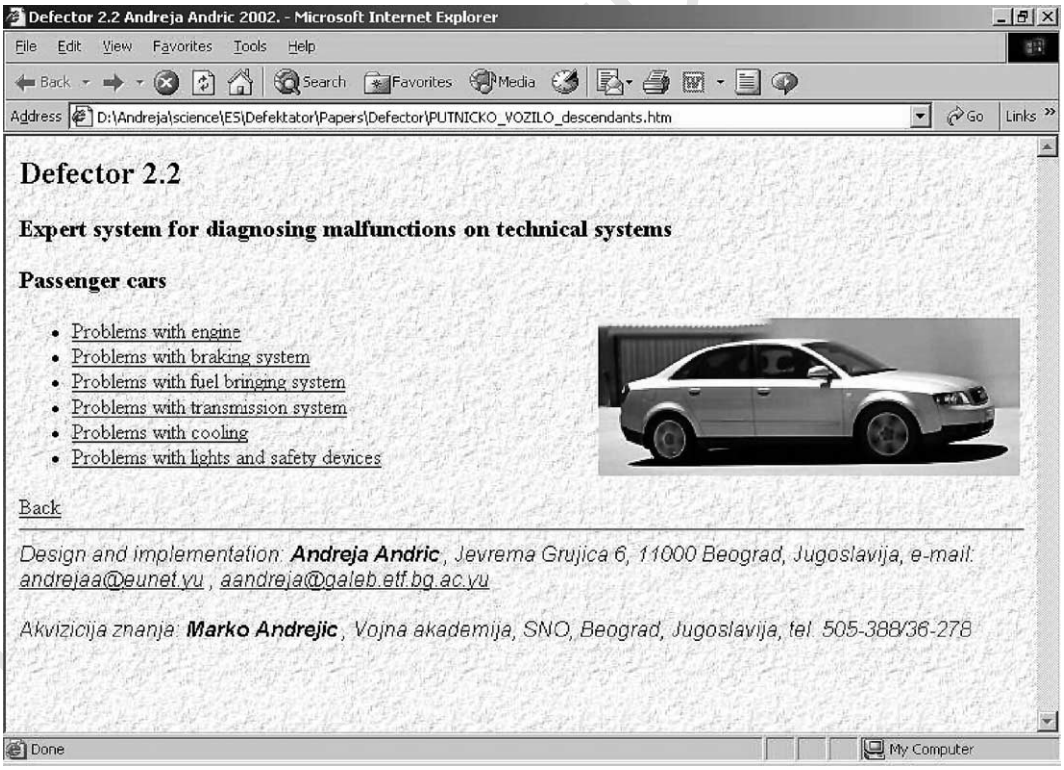

Fig. 13. A screenshot from the Defector ES.

Fig. 14. A screenshot from Travel Guide.

specification of the Jess language, were not able to create a program that would automatically check the validity of each of the knowledge base elements.

Another disadvantage of the JessGUI v.1.0 is its inability to represent uncertainty and to reason with uncertain data. Since Jess does not offer these functionalities and since the first version of JessGUI was aimed to provide support only for Jess' native options, representing and reasoning with uncertainty were not tackled until the second cycle of the JessGUI development process has begun. It is expected that the next version will be extended to overcome both this problem and the one presented in the previous paragraph.

JessGUI v.1.0 cannot import existing Jess/CLIPS documents. This limitation disables JessGUI to read, maintain, and extend existing Jess projects using advantages of JessGUI. In order to support this feature it is necessary to develop a parser for Jess/CLIPS format, but this is more difficult and time-consuming than XSLT developing. We also plan extending JessGUI's capabilities in the next version with such a converter.

8. Conclusions

The major advantages of the GUI that JessGUI provides over the UI originally supported by Jess ES Shell can be summarized as follows:

- working in graphical environment, like the one that JessGUI offers, is always preferred to using plain text editors or typing commands in DOS prompt that Jess offers as the only options;
- JessGUI facilitates the process of building Jess-based ES by enabling the users who are not familiar with details of the Jess' syntax to specify the content of the knowledge base they wish to create;
- the structure of the created knowledge base is much clearer and easier to browse;
- JessGUI supports validation of some of the knowledge-base elements immediately after their creation, thus preventing run-time errors.

Jess/JessGUI is currently used in developing both simple and complex ES within our group, as well as a teaching and learning tool in undergraduate and graduate courses on ES at our university. Its next version will be made a shareware, hoping that a large population of ES developers will contribute to its widespread use.

The next version of JessGUI, currently evolving from JessGUI v.1.0 will eliminate the need for users to learn details of Jess/CLIPS syntax. Further improvements will include the capabilities of representing uncertainty in rules and reasoning with uncertain data, as well as importing and extending existing Jess/CLIPS projects through JessGUI.

References

Alpert, S. R., Singley, M. K., & Fairweather, S. G (2000). Porting a standalone intelligent tutoring system to the web. *Proceedings of the international workshop on adaptive and intelligent web-based educational systems, Montreal, Canada* (pp. 1–11).

Andrić, A., Devedžić, V., & Andrejić, M (2003). Web-based learning environment as a passive web document. *Proceedings of the third IEEE international conference on advanced learning technologies, ICALT 2003, Athens, Greece* (pp. 22–26).

Boley, H (2001). *The rule markup language: RDF-XML data model, XML schema hierarchy, and XSL transformations.* Invited Talk, INAP2001, Tokyo, October 2001. Available at: http://www.dfki.uni-kl.de/ruleml/, last visited September 26th, 2003.

Čakić, J., & Devedžić, V. (1999). Pieces of mind: component-based software development for artificial intelligence. In I. F. Imam, Y. Kodratoff, A. Dessouki, & M. Ali (Eds.), *Multiple approaches to intelligent systems* (*Vol. 1611*) (pp. 879–888). *Lecture notes in computer science (LNCS)*, Berlin: Springer.

Carlson, D. (2001). Modeling XML applications with UML: Practical e-business applications. Boston: Addison-Wesley.

CLIPS (2003). *CLIPS, a tool for building expert systems.* Available at: http://www.ghg.net/clips/CLIPS.html, last visited September 26th, 2003.

Devedžić, V. (2002). Understanding ontological engineering. *Communications of the ACM*, *45*(4), 136–144.

Devedžić, V., & Radović, D. (1999). A framework for building intelligent manufacturing systems. *IEEE Transactions on Systems, Man, and Cybernetics. Part C. Applications and Reviews*, *29*(3), 402–419.

Durkin, J., & Durkin, J. (1998). Expert systems: Design and development. New York: Prentice Hall.

Eriksson, H. (1996). Expert systems as knowledge servers. *IEEE Expert*, *12*(2), 14–19.

Eriksson, H. (2003). Using JessTab to integrate protégé and jess. *IEEE Intelligent Systems*, *18*(2), 43–50.

Forgy, C. (1982). Rete: a fast algorithm for the many pattern/many object pattern match problem. *Artificial Intelligence*, *19*, 17–37.

Friedman-Hill, E. J (2002). *Jess, the expert system shell for the Java Platform, v. 6.1a4 User's Manual*. Available at: http://herzberg.ca.sandia.gov/jess/, last visited September 26th, 2003.

Friedman-Hill, E. J. (2003). Jess in action: Java rule-based systems. Stockholm: Manning.

Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (1995). Design patterns: Elements of reusable object-oriented software. Reading, MA: Addison-Wesley.

Giarratano, J. C. (1998). Expert systems: Principles and programming (3rd ed.). London: Brooks/Cole.

Haley Enterprise, Inc (1996). *Reasoning about Rete++. White paper*. Available at: http://www.haley.com/, last visited September 26th, 2003.

Hamada, K., et al. (1995). Hybridizing a genetic algorithm with rule-based reasoning for production planning. *IEEE Expert*, *11*(5), 60–67.

Hendler, J., Stoffel, K., Taylor, M., Rager, D., & Kettler, B (1997). *PARKA-DB: a scalable knowledge representation system—database PARKA*. Available at http://www.cs.umd.edu/projects/plus/Parka/parka-db.html, last visited September 26th, 2003.

Larman, C. (1997). Applying UML and patterns. Englewood Cliffs, NJ: Prentice Hall.

Lee, J. K., & Sohn, M. M. (2003). The eXtensible rule markup language. *Communications of the ACM*, *46*(5), 59–64.

Leff, L (2001). Automated reasoning with legal XML documents. *Proceedings of the eighth international conference on artificial intelligence and law* (pp. 215–216). New York: ACM Press

Potter, W. D., Deng, X., Li, J., Xu, M., Wei, Y., & Lappas, I. (2003). A web-based expert system for gypsy moth risk assessment. *Computers and Electronics in Agriculture*, Available at: http://webster.cs.uga.edu/~potter/dendrite/iufro-gypsy.PDF, last visited September 26th, 2003.

Sandia National Laboratories (2003). *Jess: the rule engine for the Java™ platform.* Available at: http://herzberg.ca.sandia.gov/jess/, last visited September 26th, 2003.

Šendelj, R., & Devedžić, V. (2003). Fuzzy systems based on component software. *Fuzzy Sets and Systems*, in press.

Šimić, G., & Devedžić, V. (2003). Building an intelligent system using modern Internet technologies. *Expert Systems with Applications*, *25*(3), 231–246.

Yen, J., Juang, H.-L., & MacGregor, R. (1991). Using polymorphism to improve expert systems maintainability. *IEEE Expert*, *6*(2), 48–55.

Jelena Jovanović is an MS student and a tutor at FON, School of Business Administration, University of Belgrade, Serbia and Montenegro. She has received her BS degree in information systems from FON in 2003. Her research interests include intelligent systems, knowledge representation, XML technologies, and software engineering. She is currently pursuing her MS thesis in the area of intelligent systems design. She is a member of the GOOD OLD AI research group.

Dragan Gašević is a PhD candidate at FON, School of Business Administration, University of Belgrade, Serbia and Montenegro, and a senior lecturer of computer science at the Military Academy, Belgrade, Serbia and Montenegro. He has received his BS degree in computer science from the Military Academy, Belgrade (2000), and his MS degree also in computer science from The School of Electrical Engineering, University of Belgrade (2002). His research interests include Petri nets, knowledge representation, ontologies, Semantic Web, intelligent systems, XML and related standards and technologies, and software engineering (MDA). So far, he has authored/co-authored more than 30 research papers. He is a member of the GOOD OLD AI research group, and a student member of the ACM.

Vladan Devedžić is an associate professor of computer science at the Department of Information Systems, FON, School of Business Administration, University of Belgrade, Serbia and Montenegro. He has received all of his degrees from The School of Electrical Engineering, University of Belgrade, Serbia and Montenegro (BS, 1982; MS, 1988; PhD, 1993). His main research interests include intelligent systems, knowledge representation, ontologies, Semantic Web, intelligent reasoning, software engineering, and applications of artificial intelligence techniques to education and medicine. So far, he has authored/co-authored about 180 research papers published in international and national journals and conferences. His major long-term professional goal is a continuous effort to bring close together the ideas from the broad fields of intelligent systems and software engineering. He has developed several practical intelligent systems and tools, and actively participates to several ongoing projects in industry and in academia.