Expert Systems with Applications 31 (2006) 595–606
Expert Systems with Applications
www.elsevier.com/locate/eswa

JavaDON: an open-source expert system shell

Bojan Tomić, Jelena Jovanović *, Vladan Devedžić

FON, School of Business Administration, University of Belgrade, P.O. Box 52, Jove IliĆa 154, 11000 Belgrade, Serbia and Montenegro

Abstract

The paper describes JavaDON, an open-source expert systems shell based on the OBOA framework for developing intelligent systems. The central idea of the JavaDON project was to make an easy-to-use and easy-to-extend tool for building practical expert systems. Since JavaDON is rooted in a sound theoretical framework, it is well-suited for building even complex expert system applications, both stand-alone and Web-based ones. JavaDON knowledge representation scheme supports using multimedia elements along with traditional techniques, such as rules and frames. Another important feature of JavaDON is its capability of saving knowledge bases in XML format (in addition to the shell's native format), thus making them potentially easy to interoperate with other knowledge bases on the Internet. So far, JavaDON has been used to build several practical expert systems, as well as a practical knowledge engineering tool to support both introductory and advanced university courses on expert systems. The paper presents design details of JavaDON, explains its links with the underlying OBOA framework, and shows examples of using JavaDON in expert system development. It also discusses some of the current efforts in extending JavaDON.
© 2005 Elsevier Ltd. All rights reserved.

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

1. Introduction

There is a whole range of different expert system (ES) development tools nowadays. They differ in the level of flexibility they provide in building ES and in the range of knowledge representation, reasoning, and other intelligent techniques they support. They also differ in interoperability level of knowledge bases and systems the users can develop using those tools. At the lower end are tools like *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 Inc, 1996). *Cafe Rete*, from the same manufacturer, is a Java class library that seamlessly integrates a rules engine within Java applications, servlets, EJBs, etc. At the upper end are integrated, rich ES development environments, supporting and combining several ES paradigms, as well as different mechanisms for representing and handling uncertainty, 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), (Eriksson, 1996), (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).

At the GOOD OLD AI research group (http://goodoldai.org.yu), based at the University of Belgrade, Serbia and Montenegro, there are continuous, lively activities in developing and using ES tools. We have developed a framework for building intelligent systems, called *OBOA*, and ES tools fit very well in that framework. Moreover, we have developed several practical expert systems and ES development tools, some of which are already used in practice and have been made a shareware (see Section 3 for details).

This paper presents the result of our most recent efforts—*JavaDON*, an open-source, graphical, integrated ES building tool. Although JavaDON is an academic project, the shell is capable of building even complex ES and using them either as desktop or Web applications. It still cannot rival higher-end commercial products of the same kind, but is constantly evolving and is already getting its users and supporters outside our group.

The paper is organized as follows. Section 2 defines precisely what we wanted JavaDON to provide, enable,

* Corresponding author. Tel.: +381 11 3950853; fax: +381 11 461221.
E-mail addresses: bishop@drenik.net (B. Tomić), jeljov@fon.bg.ac.yu (J. Jovanović), devedzic@fon.bg.ac.yu (V. Devedžić).

0957-4174/$ - see front matter © 2005 Elsevier Ltd. All rights reserved.
doi:10.1016/j.eswa.2005.09.085

and support. Section 3 briefly overviews the OBOA model and some other current efforts in the GOOD OLD AI group related to expert systems and ES shells. Sections 4–6 describe the overall organization of JavaDON, its built-in knowledge representation techniques and reasoning mechanisms, and details of its design. Section 7 covers one of JavaDON's most important features—using eXtensible Markup Language (XML) advantages to ensure for easy interoperability between JavaDON-based systems and other XML applications. Section 8 discusses experiences with using JavaDON so far as well as the tool's advantages and disadvantages noticed. The concluding section summarizes the paper and indicates directions for future development of JavaDON.

2. Problem statement

JavaDON project objectives include the following:

- the tool must be well engineered from the software engineering perspective;
- it must be rooted in a sound conceptual and development framework for building intelligent systems;
- it must support several knowledge representation and reasoning paradigms (including representing uncertainty and reasoning with it) and must enable their combination within a single knowledge base;
- its graphical user interface (GUI) should make building an ES easy for developers, with minimum requirements in terms of knowledge of details of the tool's internal knowledge representation format;
- it should enable building and running both desktop 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 its native knowledge bases and external Web applications;
- it should be open for further development, extensions, and integration with external intelligent tools.

JavaDON 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

JavaDON is developed after a series of other ES-related efforts within the GOOD OLD AI group. The most notable one related directly to ES building tools is *JessGUI*, a graphical user interface built on top of the *Java Expert System Shell*, or *Jess* (Friedman-Hill, 2003). Another notable ES-related research result of our group is the OBOA framework that provides a context for development of both ES and many other kinds of intelligent systems.

3.1. JessGUI

Java Expert System Shell or *Jess* (Sandia National Laboratories, 2003) is a popular, Java-based development tool for ES. Jess is simple, yet powerful enough to allow for building even industry-strength ES applications (Friedman-Hill, 2003). Its major advantage is its capability to easily integrate with external Java programs through its well-defined Java API. The API allows for controlling Jess' rule-based reasoning engine from Java programs. Jess' inference engine is mostly Rete-based (Forgy, 1982) forward chaining, but backward chaining is supported as well.

However, Jess lacks a GUI. A couple of members of the GOOD OLD AI research group, 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. 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 (Jovanović et al., 2004).

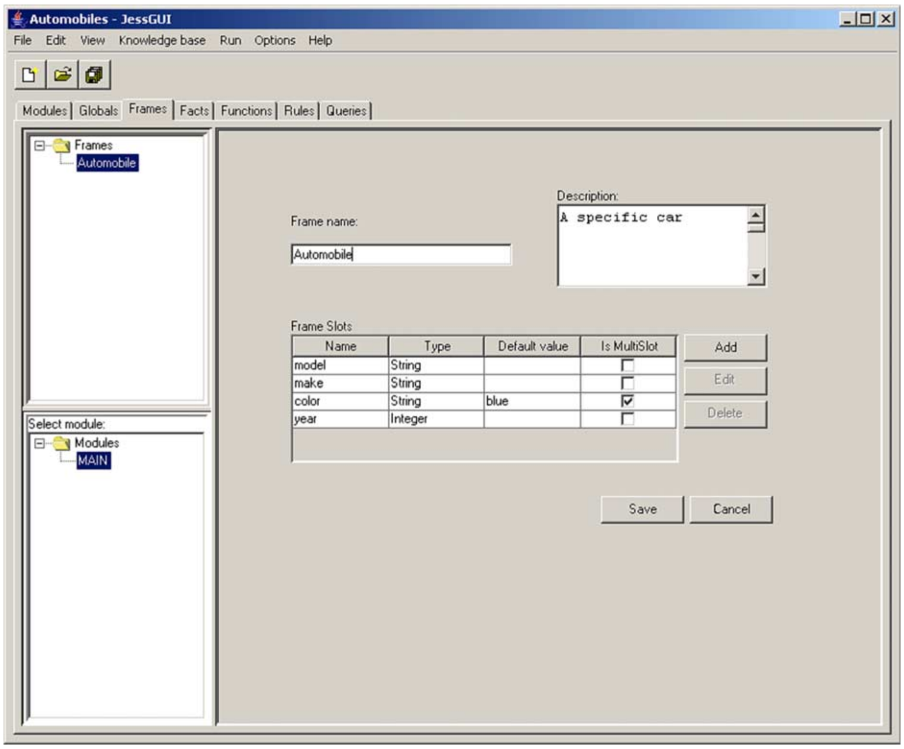
The major advantages of the GUI that JessGUI provides over the user interface originally supported by Jess can be summarized as follows:

- Working in graphical environment, like the one that JessGUI offers (Fig. 1), is always preferred to using plain text editors or typing commands in DOS prompt that Jess offers as the only options.
- JessGUI offers an alternative to learning complex syntax of the Jess (LISP-like) language. Users only need knowledge of the basic concept of the language, to be able to specify the content of the knowledge base they wish to create.

Another important advantage of JessGUI over Jess is related to interoperability. More specifically, having created an XMLSchema that defines the XML binding of Jess knowledge base, we provided JessGUI with the capability of saving knowledge bases in XML format (in addition to the original Jess/CLIPS format). That way, we enabled other Web (or non-Web) applications to access knowledge bases developed in JessGUI. However, this format cannot be used by the Jess interpreter, since it interprets only Jess/CLIPS code. Thus, we needed a way to transform the JessGUI XML format into the Jess/CLIPS code. Since XSLT is a natural solution for this kind of problem, we developed an XSLT that transforms JessGUI XML format into Jess/CLIPS format. Having developed similar XSLTs, we provided interoperability with Semantic Web tools. These XSLTs transform JessGUI's XML-based knowledge format into corresponding RDF(S) and OWL models (Jovanović & Gašević).

JessGUI v. 1.0 is now complete, and is already used in practical projects and as a teaching tool. More about this tool can be found at: http://iis.fon.bg.ac.yu/JessGUI/index.htm.

Being still a research project, JessGUI naturally has some limitations that are either in the process of elimination or are planed to be resolved in the nearest future. The most difficult one to solve is related to the import of existing Jess/CLIPS


Automobiles - JessGUI
File Edit View Knowledge base Run Options Help
Modules Globals Frames Facts Functions Rules Queries
Frames
Automobile
Select module:
Modules
MAIN
Add
Edit
Delete
Save
Cancel

Frame name: Automobile

Description: A specific car

Frame Slots

Name	Type	Default value	Is MultiSlot
model	String		☐
make	String		☐
color	String	blue	✓
year	Integer		☐

Fig. 1. Screenshot of the panel that supports frames creation.

documents. JessGUI v.1.0 is not able to read, maintain, and extend existing Jess projects. In order to support this feature it is necessary to develop a parser for Jess/CLIPS format.

We also monitor contributions of other research groups to the evolution of Jess itself. The most notable recent work in that direction is presented by Eriksson (Eriksson, 2003). He has developed a plug-in called JessTab (http://www.ida.liu.se/∼her/JessTab), which integrates Jess with Protégé-2000, a popular ontology development and knowledge acquisition tool developed at Stanford University (http://protege.stanford.edu/). JessTab enables the 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.

3.2. The OBOA framework

The OBOA framework defines five *levels of abstraction* for designing intelligent systems (Devedžić & Radović, 1999), Fig. 2a. If necessary, it is also possible to define fine-grained sublevels at each level of abstraction. Each level has associated concepts, operations, knowledge representation techniques, inference methods, knowledge acquisition tools and techniques, and development tools. They are all considered as *dimensions* along which the levels can be analyzed and instantiated, Fig. 2b. The concepts of the levels of abstraction and dimensions have been derived starting from the well-known idea of *orthogonal software architecture* from the field of software engineering (Rajlich & Silva, 1996).

Semantics of the levels of abstractions is easy to understand. In designing an intelligent system such as an ES, there are *primitives*, which are used to compose *units*, which in turn are parts of *blocks*. Blocks themselves are used to build self-contained *agents* or *systems*, which can be further integrated into more complex systems such as a distributed Web application. For getting a feeling for how the OBOA levels of abstraction correspond to some well-known concepts from the ES domain, consider the following examples. Primitives like plain text, logical expressions, attributes and numerical values are used to compose units like rules, frames, and different utility functions. These are then used as parts of

(a)

Level of abstraction	Objective	Semantics
Level	Integration	Multiple agents or systems
Level 2	System	Single agent or system
Level 3	Blocks	System building blocks
Level 4	Units	Units of blocks
Level 5	Primitives	Parts of units

(b)

Level of abstraction	Dimensions			
	D1	D2	...	DN
Level				
Level 2				
Level 3				
Level 4				
Level 5				

Fig. 2. The OBOA framework: (a) The levels of abstraction; (b) dimensions.

certain building blocks that exist in every ES, e.g. blocks of knowledge such as rule sets, and different inference strategies. At the system level, we have self-contained systems or agents like explanation planners, user modeling agents, and learning actors, all composed using different building blocks. Finally, at the integration level there are distributed AI systems, distributed learning environments, and adaptive Web-based applications (Devedžić, 2002).

It should be also noted that the borders between any two adjacent levels are not strict; they are rather approximate and 'fuzzy'. For example, a single ES can be put at the system level, as a self-contained system. However, there are equally valid arguments for putting it at the integration level if it is based on the blackboard paradigm, since it can integrate various knowledge sources. Each knowledge source in turn can be based on a different paradigm (e.g. fuzzy system, neural network, and the like) and can be developed by different tools and made to interact at a higher level.

Finally, starting from this basic framework it is easy to make further specializations and 'populate the matrix' at Fig. 2b by different items specific for a certain class of intelligent systems. For example, FuzzyOBOA framework is a specialization developed to facilitate building fuzzy systems (Šendelj & Devedžić, 2003), and GET-BITS is a specialization and customization of the OBOA framework for development of intelligent tutoring systems (Devedžić et al., 2000).

4. JavaDON requirements, architecture, and design

JavaDON has two types of users—*KnowledgeEngineers* and *EndUsers*. A *KnowledgeEngineer* creates a new ES and does all necessary modifications on existing ones. When creating a new ES, he/she collaborates with domain experts and enters the relevant domain knowledge into the knowledge base (*CreateNewProject* use case). This process usually takes a number of iterations. Modifying or extending an existing knowledge base is another typical use case (*OpenExistingProject*). The other use cases shown in Fig. 3 are self-explanatory.

An *EndUser* begins using an existing JavaDON-based ES by invoking the inference process on it, Fig. 4. He/she opens the ES, enters the problem data, and interacts with the system. The ES shows to the *EndUser* the conclusions it reaches. The interaction with the system may also include requests for explaining the inference process. An advanced *EndUser* can also select a preferred inference strategy (e.g. forward or backward chaining for rule-based inferencing; default inference strategy is forward chaining).

JavaDON architecture (Fig. 5) is in accordance with the OBOA model. Each element of the architecture belongs to one of OBOA's five levels of abstraction. In the subsequent text we give a brief overview of the architecture's elements grouped into OBOA-defined levels of abstraction.

4.1. Primitives

- *Domain*— determines the basic type and range of a value. Basic types are: string, integer, floating point and boolean.
- *Attribute*— every attribute has only one domain, and any domain can be assigned to one or more attributes.
- *Media*— represents a media element (picture, movie, etc.).
- *Action*— defines a procedure which is to be performed on a *slot* (*slot* is defined in the section 'UNITS').
- *Relation*— denotes a binary relation either between a *slot* and some value between two *slot*s (see section 'UNITS'—knowledge chunk).
- *Formula*— defines a calculation to be performed using concrete values.
- *Value*— holds a concrete single value of any type together with the certainty factor assigned to it.

4.2. Units

- *Knowledge Element*— all of the PRIMITIVES and the UNITS (no matter the complexity) are considered to be *Knowledge Element*s, since they all share common properties.
- *Kowledge Chunk*— a basic logical statement. It can be used to relate a slot with a value or with another slot. It is used as a building block for rules.
- *Frame*— similar to a class in object-oriented programming. It has properties which are called *slot*s (see *slot*). A frame

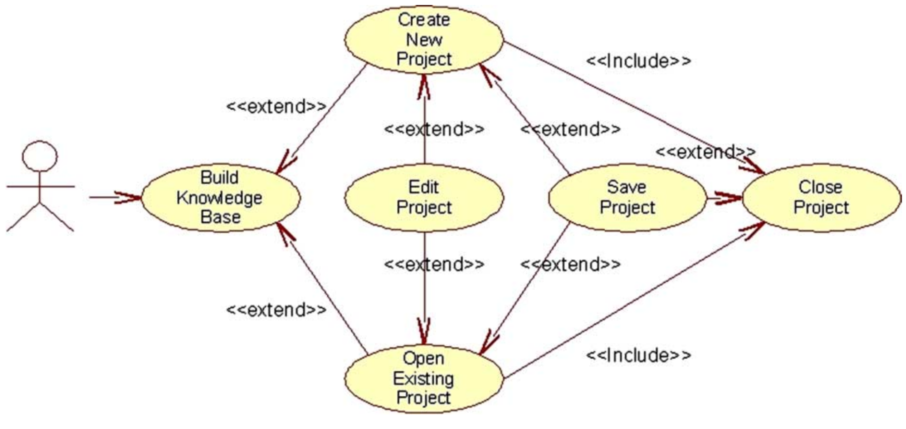


Fig. 3. KnowledgeEngineer-related use cases.

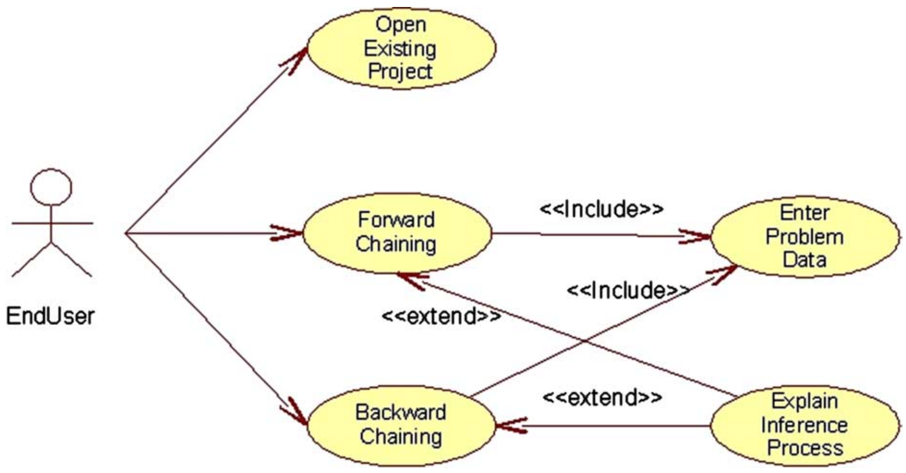


Fig. 4. EndUser-related use cases.

can inherit (‘IS A’ relation) and/or aggregate other frames (‘HAS’ relation— see *Subframe*).

- *Slot*— a link between a *frame* and an *attribute*. A slot can contain additional data, such as a question to be presented to the end user, a description, etc. Additionally, *Media* can be attached to a slot in order to better describe the associated question, or its value.
- *Subframe*— is used when a frame aggregates another frame (‘HAS’ relation).

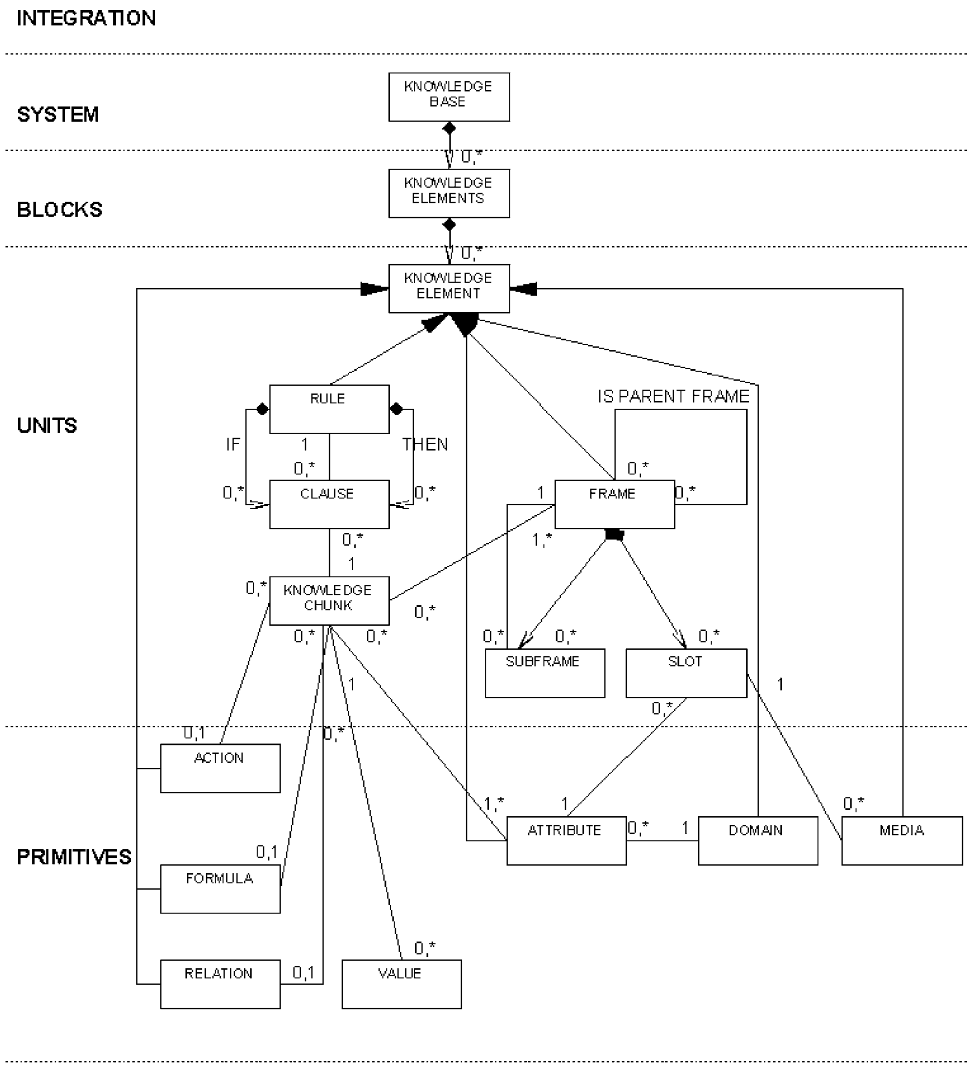


Fig. 5. JavaDON architecture in accordance with the OBOA model.

- *Rule*—a form of representing knowledge using IF/THEN statements (clauses). Each rule has its prerequisites (one or more IF clauses) which have to be satisfied in order for the THEN part to be true. If a rule has multiple IF clauses, they can be internally related with the 'OR' or the 'AND' logical operator. That means that one ('OR') or all ('AND') of the IF clauses have to be evaluated as true before THEN clauses can be proclaimed true.
- *Clause*—a link between a *Knowledge Chunk* and a *Rule.*

4.3. Blocks

Knowledge Elements—all *Knowledge Elements* can be grouped. These groups contain only *Knowledge Elements* of the same type (rules, frames...).

4.4. System

Knowledge Base—represents all knowledge of a certain domain an ES contains. Knowledge Base consists of several knowledge element groups *(Knowledge Elements).*

4.5. Integration

Currently, JavaDON does not support other ways of integrating several knowledge bases except for merging them in a single knowledge base. Integration with other intelligent systems can be achieved through JavaDON-s API, databases, or files it produces (see Section 6 for details).

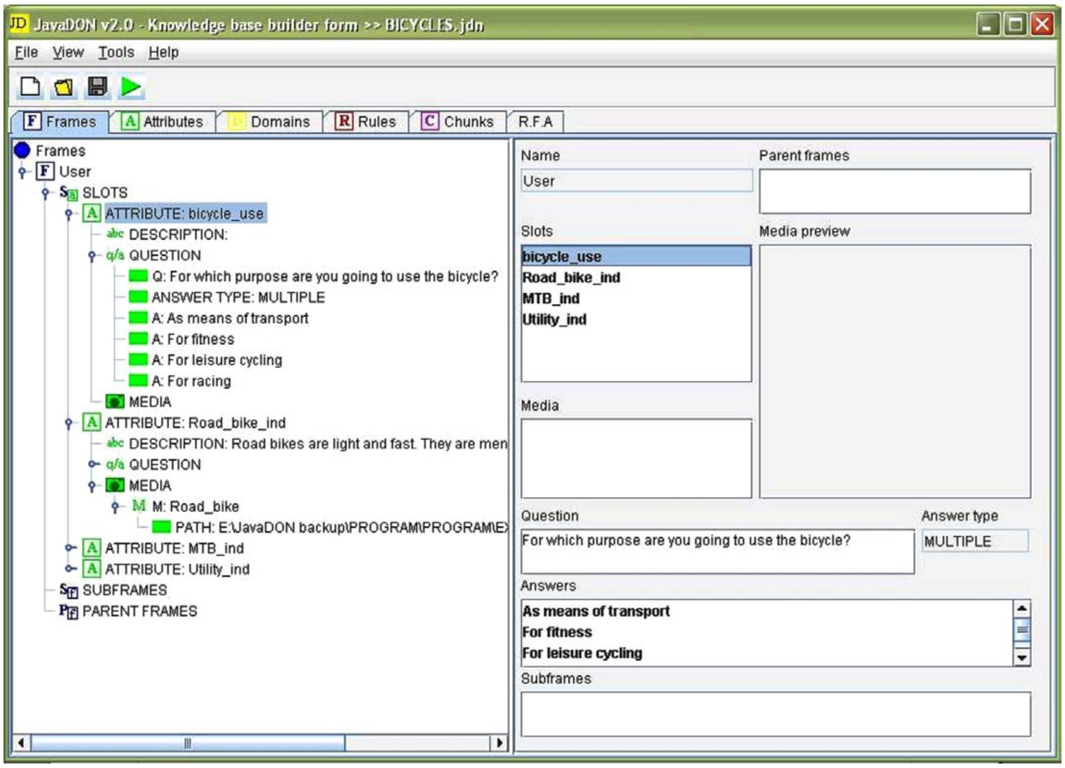
From this point forward, a simple example will be used to describe main JavaDON features. The example is a part of an ES that provides support in choosing an appropriate type of bicycle.

5. Knowledge representation in JavaDON

JavaDON implements all knowledge representation techniques defined in the OBOA meta-model, namely: frames, rules and O-A-V triplets. All three are used simultaneously, and with some modification (i.e. a frame cannot contain procedures as the OBOA specifies).

Considering our example ES, Figs. 6 and 7 illustrate how a frame is represented in the JavaDON's GUI.

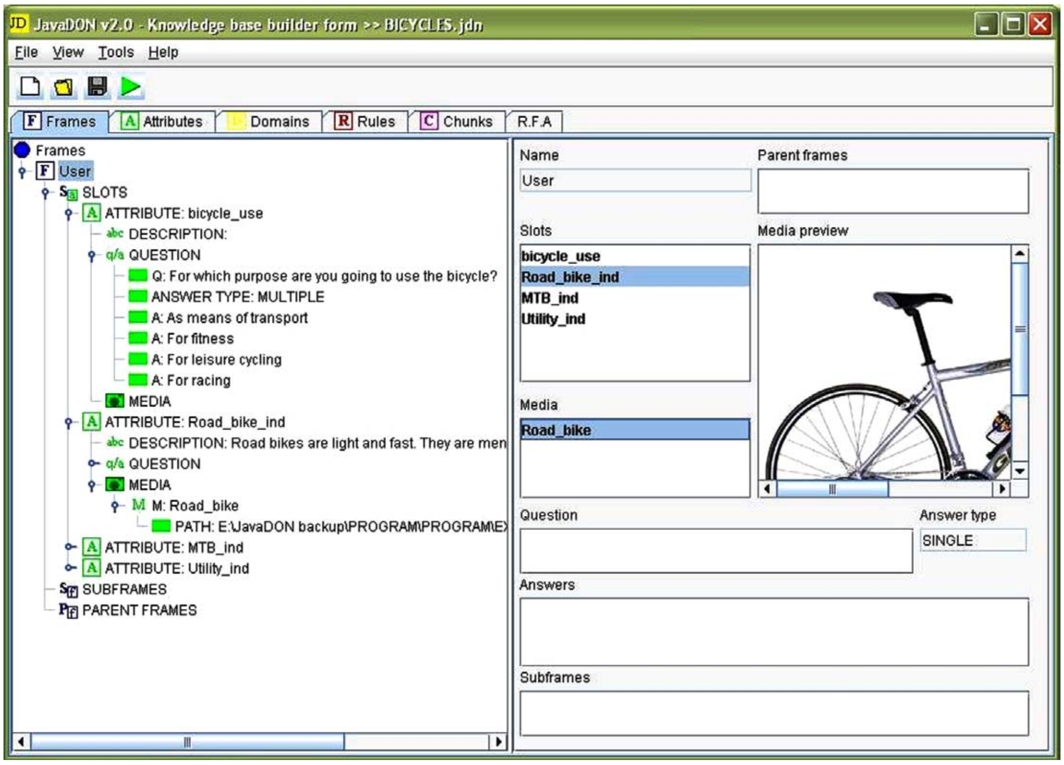
Fig. 6 shows a screenshot of a frame with its contents. At the left half of the picture, one can see a tree structure. The root of the tree is node 'Frames' which represents all frames in the ES. All subtrees that branch directly from the root represent frames. In the presented screenshot, there is only one frame called 'User'. In JavaDON, a frame can have slots, subframes and parent frames. Each slot is represented as a subtree with the root node 'SLOTS'. The frame in our example ('User') has only four slots, while two nodes ('SUBFRAMES' and 'PARENT FRAMES') have no child nodes. A slot is a connection between a frame and an attribute, so this information is shown in the node 'ATTRIBUTE'. A slot can have a description ('DESCRIPTION'), one or more media files


JavaDON v2.0 - Knowledge base builder form >> BICYCLES.jdn
File View Tools Help
Frames Attributes Domains Rules Chunks R.F.A

Frames
User
SLOTS
ATTRIBUTE: bicycle_use
DESCRIPTION:
QUESTION
Q: For which purpose are you going to use the bicycle?
ANSWER TYPE: MULTIPLE
A: As means of transport
A: For fitness
A: For leisure cycling
A: For racing
MEDIA
ATTRIBUTE: Road_bike_ind
DESCRIPTION: Road bikes are light and fast. They are men
QUESTION
MEDIA
M: Road_bike
PATH: E:\JavaDON backup\PROGRAM\PROGRAME
ATTRIBUTE: MTB_ind
ATTRIBUTE: Utility_ind
SUBFRAMES
PARENT FRAMES

Name: User
Parent frames:
Slots: bicycle_use, Road_bike_ind, MTB_ind, Utility_ind
Media preview:
Media:
Question: For which purpose are you going to use the bicycle?
Answer type: MULTIPLE
Answers: As means of transport, For fitness, For leisure cycling
Subframes:

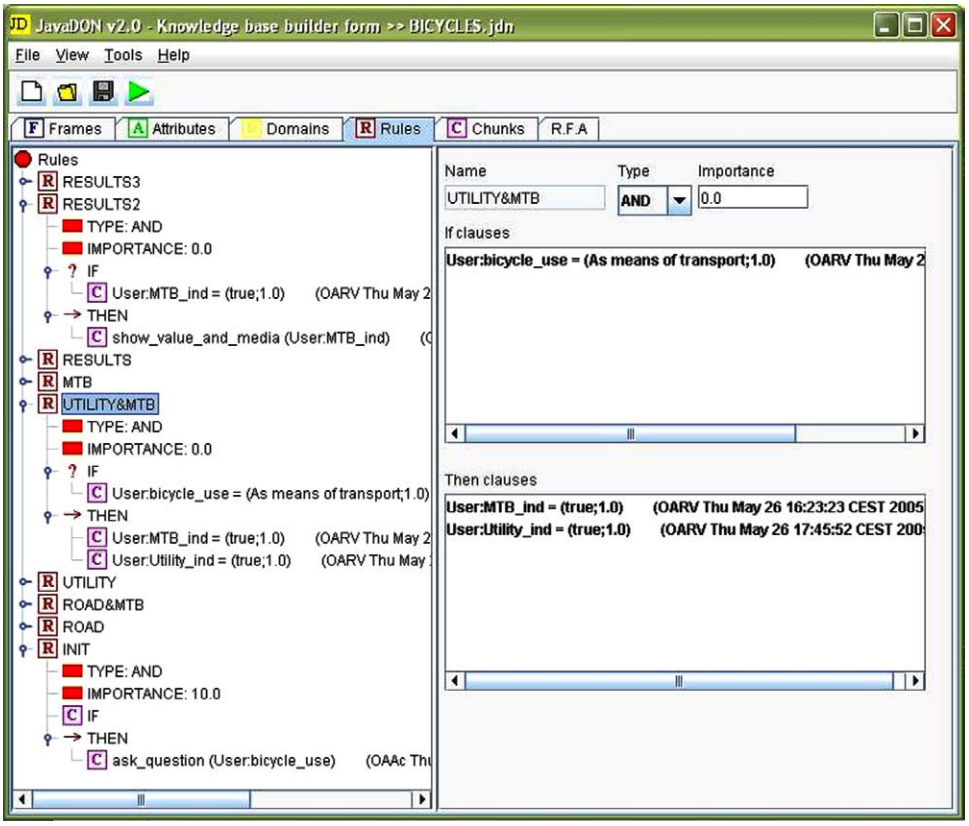
Fig. 6. A JavaDON screenshot: frame 'User' with its slot 'bicycle_use' shown.


JavaDON v2.0 - Knowledge base builder form >> BICYCLES.jdn
File View Tools Help
Frames Attributes Domains Rules Chunks R.F.A

```
Frames
  F User
    SLOTS
      A ATTRIBUTE: bicycle_use
        DESCRIPTION:
        QUESTION
          Q: For which purpose are you going to use the bicycle?
          ANSWER TYPE: MULTIPLE
          A: As means of transport
          A: For fitness
          A: For leisure cycling
          A: For racing
        MEDIA
      A ATTRIBUTE: Road_bike_ind
        DESCRIPTION: Road bikes are light and fast. They are men
        QUESTION
        MEDIA
          M: Road_bike
            PATH: E:\JavaDON backup\PROGRAM\PROGRAM\E
      A ATTRIBUTE: MTB_ind
      A ATTRIBUTE: Utility_ind
    SUBFRAMES
    PARENT FRAMES
```

Name: User
Parent frames:
Slots: bicycle_use, Road_bike_ind, MTB_ind, Utility_ind
Media preview
Media: Road_bike
Question:
Answer type: SINGLE
Answers:
Subframes:

Fig. 7. JavaDON screenshot: frame ‘User’ with its slot ‘Road_bike_ind’ shown.


JavaDON v2.0 - Knowledge base builder form >> BICYCLES.jdn
File View Tools Help
Frames Attributes Domains Rules Chunks R.F.A

```
Rules
  R RESULTS3
  R RESULTS2
    TYPE: AND
    IMPORTANCE: 0.0
    ? IF
      C User:MTB_ind = (true;1.0)     (OARV Thu May 2
    → THEN
      C show_value_and_media (User:MTB_ind)     (C
  R RESULTS
  R MTB
  R UTILITY&MTB
    TYPE: AND
    IMPORTANCE: 0.0
    ? IF
      C User:bicycle_use = (As means of transport;1.0)
    → THEN
      C User:MTB_ind = (true;1.0)     (OARV Thu May 2
      C User:Utility_ind = (true;1.0)     (OARV Thu May
  R UTILITY
  R ROAD&MTB
  R ROAD
  R INIT
    TYPE: AND
    IMPORTANCE: 10.0
    C IF
    → THEN
      C ask_question (User:bicycle_use)     (OAAc Thu
```

Name: UTILITY&MTB   Type: AND   Importance: 0.0
If clauses:
User:bicycle_use = (As means of transport;1.0)   (OARV Thu May 2
Then clauses:
User:MTB_ind = (true;1.0)   (OARV Thu May 26 16:23:23 CEST 2005
User:Utility_ind = (true;1.0)   (OARV Thu May 26 17:45:52 CEST 200

Fig. 8. JavaDON screenshot: rule representation.

(‘MEDIA’) and a question and answers (‘QUESTION’) associated with it. This description and/or media files can be shown to the end user to better explain the solution the ES has produced. The question is used to gather information about the value of the slot, if the value is to be supplied by the end user. It can have two or more predefined answers (with the choice of single or multiple selection), or it can be left to the user to supply one or more answers. For example, slot ‘bicycle_use’ shown in the Fig. 6, has a question associated with it. Accordingly, when ES needs information about the way the end user is going to use his/her bicycle, it will ask him/her “For which purpose are you going to use the bicycle?” and present a list of four answers to choose from (with multiple selection option): “As means of transport”, “For fitness”, “For leisure cycling”, “For racing”. Media files can be presented to the end user together with the question, in order to describe it better. On the right side of the screen, the selected frame and its slots are shown using list boxes, text boxes, etc.

Fig. 7 shows the screenshot of the same frame, but with different slot selected (“Road_bike_ind”). Since its value is determined by the ES (because it is a goal fact) it has no question associated with it. Instead, it has a media file (picture of a bicycle on the right side of the screenshot) and a description associated with it.

In JavaDON, a rule consists of four elements: type, importance, IF and THEN clauses (Fig. 8). The type of a rule determines whether the rule’s IF clauses are related with each other using the logical “and” or the logical “or” operation. Importance determines which rule will be executed in case there are more rules that can be executed (more on reasoning in the next chapter). The IF and THEN clauses are the meeting point between frames and rules. Each clause uses one or more frame slots in order to build a logic expression. Let us look at the rule “UTILITY&MTB”. The rule states that if the “User”-s slot “bicycle_use” has the value of “As means of transport” (IF clause), then slots “MTB_ind” and “Utility_ind” will get the value “true” assigned to them (THEN clauses). That means that if the end user wishes to use the bicycle as means of transport, that the optimal solution for him/her is either the ‘utility bicycle’ or the ‘mountain bicycle’.

Actions can only be performed through rules. Rule “INIT” is an example of that. Since it has no IF clauses, it can be executed at the moment of starting the ES. Furthermore, since it has the highest possible level of importance (Giarratano, 1998), it surely will be executed first. That means that its THEN clause will be executed when the ES starts, prompting the end user to answer the question associated with slot “bicycle_use” from the frame “User”—”For which purpose are you going to use the bicycle?”.

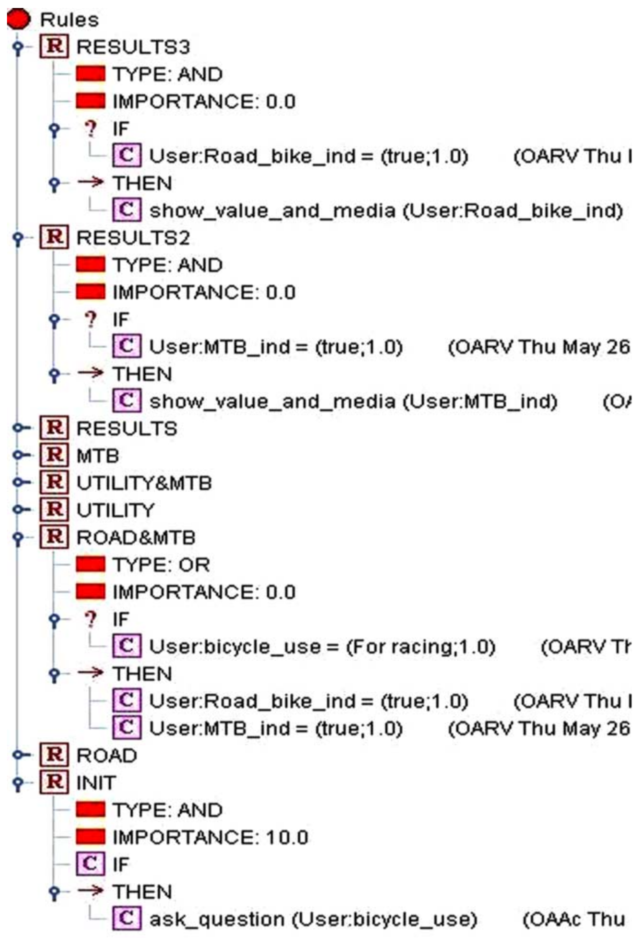
6. Reasoning in JavaDON

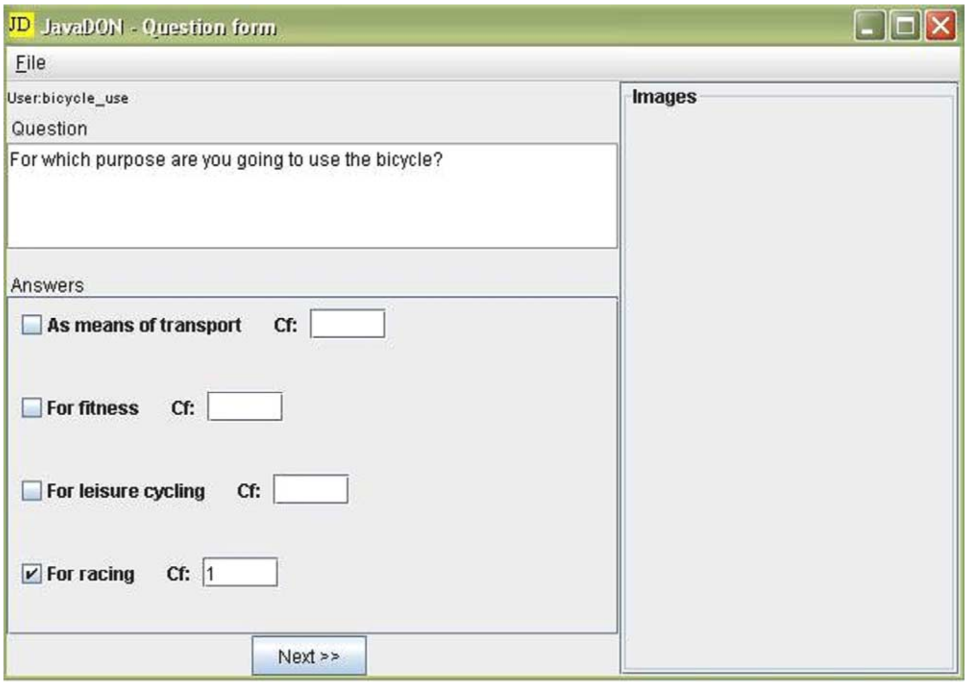
All of the reasoning in JavaDON is done by using the forward chaining inference technique. Conflict resolving is based on two techniques: ‘highest priority rule’ and ‘rule can be fired only once’. Currently we are working on the implementation of the backward chaining technique.

We explain the process of inferencing in JavaDON through our example ES (all of the rules it consists of are shown in Fig. 9). Only the rules relevant for the example are fully expanded in the tree view.

When the ES starts, it initializes the inference process. In each round of the process, a set of relevant rules is created. This set consists of all the rules that have their IF clauses satisfied and have not yet been fired. Then, only one rule is chosen from the set and fired (its THEN premises are added to the working memory), which concludes the round. This cycle continues until the set of relevant rules is empty, or the user-defined ‘stop’ action is reached. In case the set consists of more rules, which is very probable by the way, the rule with the highest priority is chosen to be fired (‘highest priority rule’ technique). Also, a rule can be fired only once during the whole inference cycle (‘rule can be fired only once’ technique).

Correlating with our example, we can see that rule “INIT” is the first to get fired when the example ES starts. It is the only rule with IF part satisfied (it has no premises), and therefore the only one in the set of relevant rules. That means that action in its then part (“ask_question(User:bicycle_use)”) will be executed. The screen presented to the end user is shown in Fig. 10.


```
Rules
  RESULTS3
    TYPE: AND
    IMPORTANCE: 0.0
    ? IF
      C User:Road_bike_ind = (true;1.0)      (OARV Thu I
    -> THEN
      C show_value_and_media (User:Road_bike_ind)
  RESULTS2
    TYPE: AND
    IMPORTANCE: 0.0
    ? IF
      C User:MTB_ind = (true;1.0)      (OARV Thu May 26
    -> THEN
      C show_value_and_media (User:MTB_ind)      (O/
  RESULTS
  MTB
  UTILITY&MTB
  UTILITY
  ROAD&MTB
    TYPE: OR
    IMPORTANCE: 0.0
    ? IF
      C User:bicycle_use = (For racing;1.0)      (OARV Th
    -> THEN
      C User:Road_bike_ind = (true;1.0)      (OARV Thu I
      C User:MTB_ind = (true;1.0)      (OARV Thu May 26
  ROAD
  INIT
    TYPE: AND
    IMPORTANCE: 10.0
    C IF
    -> THEN
      C ask_question (User:bicycle_use)      (OAAc Thu
```
Fig. 9. Rules in the example ES (part of a JavaDON screenshot).



User:bicycle_use

Question

For which purpose are you going to use the bicycle?

Answers

- [ ] As means of transport Cf:
- [ ] For fitness Cf:
- [ ] For leisure cycling Cf:
- [✓] For racing Cf: 1

Fig. 10. Question presented to the end user in the example ES (JavaDON screenshot).

Let's say that the end user wishes to use the bicycle only for racing. He/she checks the appropriate answer and presses the button 'Next> >', and a new round of the inference cycle begins. The set of relevant rules has only one element—rule 'Road&MTB' (Fig. 9). When this rule is fired, its THEN clauses are put in the working memory. Practically, this means that the indicators representing the 'road bike' and the 'mountain bike' are set to true (slots "Road_bike_ind" and "MTB_ind" from the frame "User"), thus appointing these two types of bicycles as optimal for the end user. In the next two rounds, "RESULT2" and "RESULT3" rules are fired, executing actions that present the values of the aforementioned slots, together with the associated media files, to the end user (Fig. 11). After this, the newly formed set of relevant rules is empty which means that the inference process ends.

7. Interoperability issues

The first version of JavaDON supported storage of developed knowledge bases in XML format compliant to the predefined XML Schema. Furthermore, procedural knowledge, represented in the form of rules, could have been saved in Rule Markup Language (RuleML).[1] A set of XSLTs was also implemented in order to enable transformation of JavaDON's native XML format into JessGUI, RDF(S) and OWL XML formats (Jovanović & Gašević). However, since this old version of the tool implemented only a small part of the features covered by the new version, the practical use of the existing XML Schema as well as developed XSLTs was very limited.

Two members of our research team (Dražen Krsmanović and Nebojša Antić) worked on JavaDON's support for XML serialization. As the result of their efforts, the current version of JavaDON is equipped with:

- An XML Schema covering all features of the OBOA model;
- XSLTs that enable bidirectional conversions between JavaDON's native XML format and RuleML format, as well as between JavaDON's and JessGUI's native XML formats.

A part of the developed XML Schema is shown in the Fig. 12. The Fig. presents the *Knowledge Element* unit (see Section 4) and exposes details of the structure of one of the *Knowledge Element* types (*Frame* type). Currently, efforts are made to implement RDF and OWL compatibility.

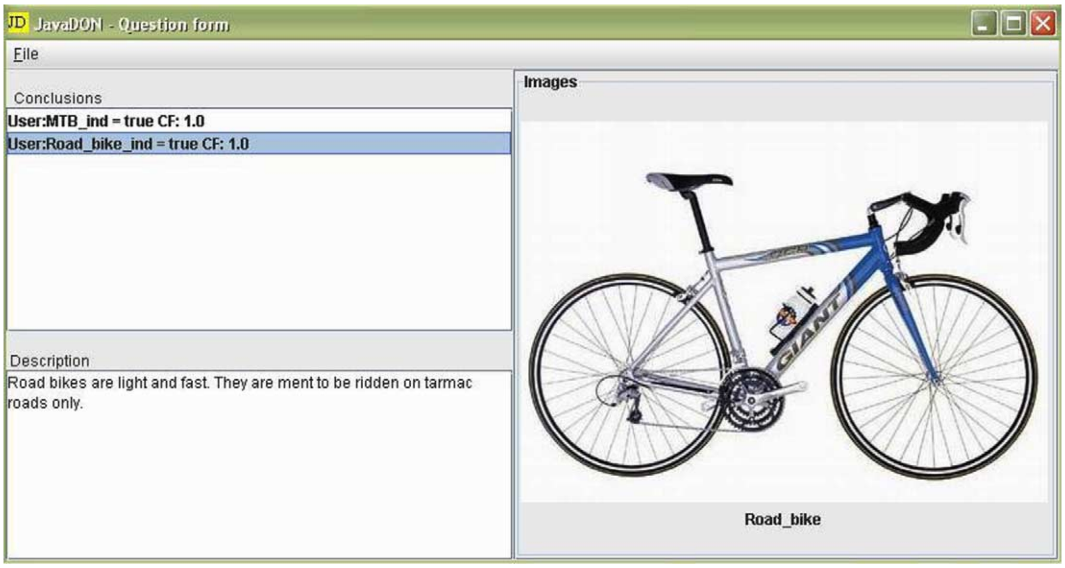
Furthermore, it is possible to use JavaDON as a part of any Java-based application, through its API. The API consists of standard Java methods that can be called in order to build or execute an ES.

8. Applications and discussion

In this section we give an overview of ES that have been created using JavaDON Shell. These ES have been developed by undergraduate students, members of our research group, as part of their final thesis.

Bicycle expert system. This ES helps a potential buyer to determine which type of bicycle he/she should buy. The solutions it generates are determined by the way the buyer is

[1] RuleML is an ongoing effort of The Rule Markup Initiative (http://www.dfki.uni-kl.de/ruleml/) to define a shared language, that would permit both forward (bottom-up) and backward (top-down) rules in XML for deduction, rewriting, and further inferential-transformational tasks (Boley, 2001), (Lee and Sohn, 2003).


JavaDON - Question form

File

Conclusions

User:MTB_ind = true CF: 1.0

User:Road_bike_ind = true CF: 1.0

Description

Road bikes are light and fast. They are ment to be ridden on tarmac roads only.

Images

Road_bike

Fig. 11. Conclusions represented to the end user by the example ES.

going to use the bicycle. Part of this ES is used in the previous sections of the paper to explain JavaDON features.

Travel GuidES was made by Vera Vasiljevic. It is an ES that provides assistance in choosing an appropriate location and accommodation for a holiday. It consists of several independent ES specialized in certain types of vacation—summer, winter, etc. Since the ES is supposed to run on the web, an applet was made to provide interface to end users (Fig. 13).

Lora ES was made by Marija Suljamčević. It simplifies decision making regarding the best suited hairstyle for a user. The input data are face and hair features such as: face shape, hair color, hair thickness, etc.

JavaDON has already proved as a practical knowledge engineering tool in both introductory and advanced university courses on expert systems. During this spring semester, we used JavaDON as a teaching/learning tool in courses such as Expert Systems, Intelligent Information Systems and the like. Our intention was to provide students with a powerful, yet simple to use tool that would help them understand the basics of knowledge representation and reasoning techniques. Students initially used it to develop simple ES in the labs, and later to do more advanced projects. We have already conducted an informal evaluation of JavaDON with the students. They were asked to give their view of the usefulness of JavaDON tool for acquiring practical knowledge on ES development. Along with positive feelings about JavaDON, the students have also provided critical opinions that were extremely useful for further improving JavaDON. Students were also asked to compare JavaDON tool with JessGUI. The majority of students preferred JavaDON over JessGUI, stressing that it is more comfortable to work with. As we see it, the primary reason for such students' attitude should be traced to the underlying knowledge models of these tools. As we explained, JavaDON is built on top of an object-oriented

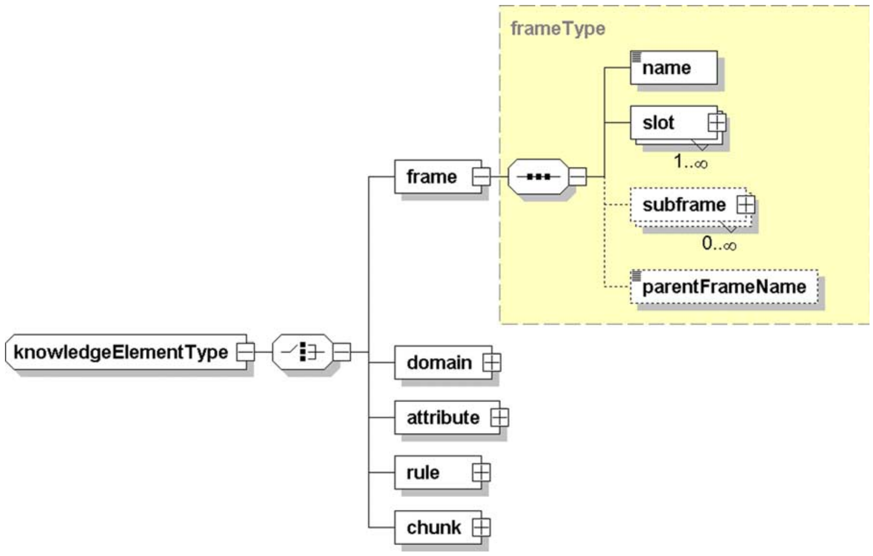
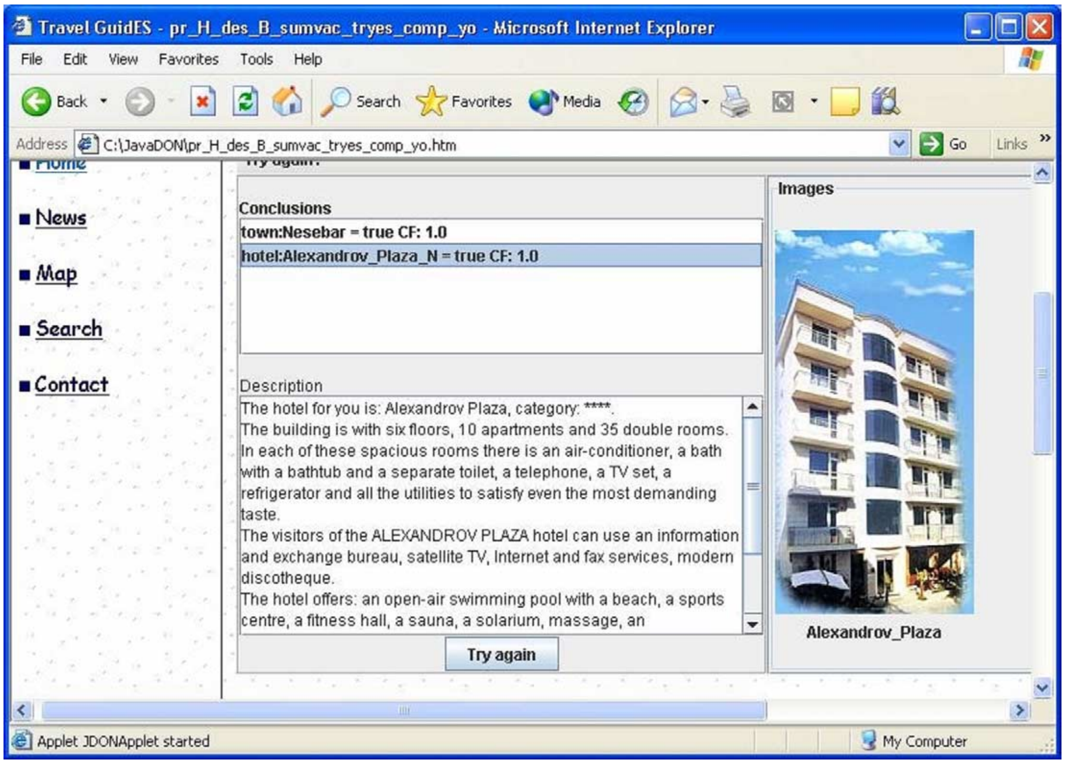


Fig. 12. A part of JavaDON XML Schema: Knowledge Element unit.


Travel GuidES - pr_H_des_B_sumvac_tryes_comp_yo - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Address C:\JavaDON\pr_H_des_B_sumvac_tryes_comp_yo.htm
Go Links
Home
News
Map
Search
Contact
Images
Alexandrov_Plaza
Try again
Applet JDONApplet started
My Computer

Conclusions

town:Nesebar = true CF: 1.0

hotel:Alexandrov_Plaza_N = true CF: 1.0

Description

The hotel for you is: Alexandrov Plaza, category: ****.
The building is with six floors, 10 apartments and 35 double rooms.
In each of these spacious rooms there is an air-conditioner, a bath with a bathtub and a separate toilet, a telephone, a TV set, a refrigerator and all the utilities to satisfy even the most demanding taste.
The visitors of the ALEXANDROV PLAZA hotel can use an information and exchange bureau, satellite TV, Internet and fax services, modern discotheque.
The hotel offers: an open-air swimming pool with a beach, a sports centre, a fitness hall, a sauna, a solarium, massage, an

Fig. 13. Travel GuidES screenshot.

(OO) knowledge base model and our students are taught to think in OO terms from the first year of their studies. Therefore, it was expected that they will more easily accept a tool compliant with their mindset, than the one built on top of an unfamiliar and rather complex knowledge-base model. The students were additionally asked to compare JavaDON with KAPPA-PC ES shell (http://www.intellicorp.com/kappa-pc/) that is also built on top of an OO model but is a commercial product and hence more complex and powerful. Students were almost unanimous in their opinions: JavaDON is better in the early learning phase, when fundamentals of ES development techniques should be acquired. However, KAPPA-PC was perceived as a better solution for developing more complex ES. Again, this feedback was exactly in accordance with our expectations.

The new version of JavaDON enables users to build complex ESs without knowledge of any specific programming language. It is simple and intuitive. The tool enables building and running both desktop and Web-based ES applications: desktop applications are run through JavaDON's ES launcher, whereas an applet is available for the web-based ones. JavaDON is developed using the three-tier architecture and design patterns, so it can be easily maintained and extended. Being developed in Java programming language, JavaDON can be easily integrated with external intelligent tools through its Java API.

JavaDON is rooted in OBOA framework—a sound conceptual framework for building intelligent systems. It supports several knowledge representation techniques (frames, rules and OAV triplets) and uses certainty factors for representing uncertainty and reasoning with it. Furthermore, an XMLSchema defining XML binding of JavaDON native knowledge bases has been developed, hence improving JavaDON's interoperability with other intelligent tools. A pair of XSLTs has already been built enabling interoperability with JessGUI. Furthermore, we are currently developing an additional set of XSLTs that will make JavaDON interoperable with Semantic Web tools (i.e. tools based on RDF(S) and OWL languages). As stated at the beginning of this section, the tool has already undergone a thorough testing through development of a number of simple practical applications.

Still, a few JavaDON features were noted as impractical, and have to be improved. First, paths to the media files that ES uses are saved as directory paths. Instead, they should be in form of classpaths for higher portability level. Next, certainty factors must be included whenever the end user is entering data. This proved to be unnecessary in some cases, so it should be made optional. Finally, when the end user is entering problem data (as answers), he/she can only move forward through the questions or restart the ES. This is especially impractical when the ES contains a lot of questions, or when the end user changes his/her mind about some of the given answers. In order to prevent this problem from occurring, a 'truth maintenance' system must be implemented.

9. Conclusions

The experience we had with using JavaDON in practice so far allows us to summarize its major advantages as follows:

- the tool is rooted in the OBOA framework, which provides both a stable theoretical foundation and a firm backbone for future extensions;
- due to its partial support for multimedia knowledge representation, the shell enables developing ES with rich graphical interface, although it does not integrate a separate user-interface development tool;

the structure of the created knowledge base is clear and easy to browse;

- JavaDON facilitates introducing the process of building ES to inexperienced users, due to its highly intuitive graphical user interface; it allows the users to specify the content of the knowledge base they wish to create without the need to cover a specific knowledge representation language first;
- it is an open-source integrated tool, and is easy to extend with new knowledge representation and inference techniques;
- its capability of creating knowledge bases in the XML format increases its interoperability with other intelligent systems and tools;
- expert systems developed with JavaDON can be either desktop or Web applications;
- JavaDON supports validation of some of the knowledge-base elements immediately after their creation, thus preventing run-time errors.

JavaDON 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 current version is made a shareware (http://iis.fon.bg.ac.yu/DodatniMaterijali/JavaDON.zip), hoping that a large population of ES developers will contribute to its widespread use.

The next version of JavaDON will have a slightly modified user interface, in order to accommodate the comments and feedback we received from the users so far. Further improvements will include better support for using certainty factors and other techniques for representing uncertainty in rules and reasoning with uncertain data, as well as support for importing and extending existing knowledge bases developed with other tools.

References

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.

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.

Devedžić, V., Jerinić, L., & Radović, D. (2000). The GET-BITS model of intelligent tutoring systems. *Journal of Interactive Learning Research (JILR)*, *11*(3/4), 411–434 (Special issue on intelligent systems/tools in training and lifelong learning).

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. (2003). *Jess in action: Java rule-based systems*. Stockholm: Manning.

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 May 16th, 2005.

Jovanović, J., & Gašević, D. (forthcoming). XML/XSLT-based knowledge sharing. *Expert Systems with Applications*, 29(3).

Jovanović, J., Gašević, D., & Devedžić, V. (2004). A GUI for Jess. *Expert Systems with Applications*, *26*(4), 625–637.

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

Rajlich, V., & Silva, J. H. (1996). Evolution and reuse of orthogonal architecture. *IEEE Transactions on Software Engineering*, *22*(2), 153–157.

Sandia National Laboratories. (2003). Jess: The rule engine for the JavaTM 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*, *141*(3), 487–504.