WebExplain: A UPML Extension to Support the Development of Explanations on the Web for Knowledge-Based Systems

Vládia Pinheiro[1], Vasco Furtado[1], Paulo Pinheiro da Silva [2], Deborah L. McGuinness[3]
[1] University of Fortaleza, Fortaleza, Ceará, Brazil
[2]University of Texas at El Paso, TX, USA
[3] Knowledge Systems Laboratory, Stanford University, Stanford, CA, USA

***Abstract*—** Knowledge-based systems (KBS) should be able to explain their results to improve the understanding and credibility of their answers by users. However, most KBS explanation components cannot be easily reused by other applications, thus increasing the effort of implementing KBSs with explanation capabilities. In this paper we present WebExplain, an extension to Unified Problem-Solving Method Description Language (UPML), a KBS development framework. WebExplain is integrated with UPML generic components and can be easily reused during the development of other problem-solving methods and KBSs. WebExplain uses the Inference Web for enabling proof and explanation interoperability between distributed applications. We exemplify our approach by describing WebExplain's use in the development of a problem-solving method and a KBS with explanation capabilities.

1. INTRODUCTION

Problem-solving methods (PSMs) describe the reasoning steps and the knowledge roles used during problem-solving processes, regardless of the problem domains. PSMs can be reused by many applications. For example, a single PSM may be used to guide a knowledge acquisition process as well as to describe the design of a knowledge-based system [3].

KBS users need to understand the manner in which solutions provided by a KBS were produced. They then need explanations that describe the PSM flow and the inference steps followed by the KBS in order to produce a result. Despite the development of explanation components by some knowledge engineering solutions (e.g., [1,2,15,17,18]), no known component is able to explain answers from general PSM implementations. Some KBS can generate explanations based on generic tasks or meta-rules, but we are unaware of any explanation approach that provides a systematic way to generate explanations for PSM implementations or that can be easily reused for the development of other KBSs, thus reducing the effort of implementing explanatory KBSs. This paper describes an extension of the Unified Problem-Solving Method Description Language (UPML) [4]—, a KBS development framework, by providing a reusable component with explanation capabilities for KBSs. This component, called WebExplain, has two main characteristics: (i) It is integrated with UPML generic components: PSM, Task and Ontologies (i.e., Method Ontology and Task Ontology). In UPML, a PSM, a Task and their ontologies can be reused in several domains in order to develop several and different KBSs. Hence, we are leveraging UPML and benefiting from the reusability of the UPML infrastructure, thereby reducing the cost of development for Explainable KBS; (ii) It generates justifications for the KBS results in the Proof Markup Language (PML) [11]. PML is the proof format of the Inference Web (IW), an infrastructure for Web explanations enabling applications to generate portable and distributed explanations for any of their answers [8]. PML has an OWL-DL encoding [10], and is thus compatible with semantic web applications in XML, RDF, and OWL. It includes content such as sources, inference rules, inference steps, and conclusions. Thus, proofs in PML can be shared with other distributed applications on the Web, besides using the IW infrastructure and tools to abstract proofs into explanations and to present them to users. KBSs are increasingly being deployed in heterogeneous environments such as the Web—for example, Web Services, and sharing information with other applications. Hence, there is a need for interoperable KBS responses and explanations, like PML.

UPML-based KBS development makes reasoning processes explicit by implementing PSM as part of the applications. Thus, the capability of accessing PSMs at execution time can be leveraged to explain PSM answers at the reasoning level—the strategic level. With the help of UPML patterns, one can enhance the quality of explanations by abstracting away task-specific reasoning steps from proofs and by keeping relevant information for response understanding. Explanations about the domain knowledge are generated from the KBS's inference engine that chains the domain rules.

We exemplify our approach by describing how to have reusability in both the development of PSM and the development of Explainable UPML-based KBS.

2. BACKGROUND KNOWLEDGE

Our approach for Explainable KBS integrates UPML and the IW frameworks.

2.1. Inference Web

Inference Web (IW) [8,9] is a framework for explaining reasoning tasks by storing, exchanging, combining, abstracting, annotating, comparing, and rendering answer justifications[1] provided by reasoners embedded in applications. IW justifications identify the KBS reasoning steps used to derive answers from input information. In addition to a language for answer justification, IW provides an infrastructure that includes: an extensible web-based registry containing details on information sources, reasoners, languages, and rewrite rules; a justification abstractor, and explanation browser. The browser is used to support navigation and presentations of answer justifications. The explainer is used to abstract machine-level justifications into human-level explanations.

PML is the IW justification specification language and includes two major components for building proof trees: inference steps and node sets. A justification can then be defined as a tree of inference steps explaining the process of deriving answers, which are the final conclusions of a justification. Node sets represent both the antecedents and conclusions of inference steps. In other words, an inference step is the application of a single inference rule over a set of antecedents (encoded as node set conclusions) and deriving a consequent (also encoded as a node set conclusion). Each inference step contains pointers to the inference rule and variable mappings used.

2.2. UPML

The UPML framework [4] supports KBS modeling from reusable components, adapters, development guidelines, a description language, and tools. Distinct KBS software components are described by the UPML architecture:

- PSM component that defines the control structure responsible for the coordination of subtasks, i.e., the definition of the subtask execution order;
- Task component that defines the problem that should be solved by the KBS. Subtasks executed by the PSM component are also task components that implement the procedure in order to solve one part of the overall problem. Normally, the subtasks are implemented through an inference engine that executes the rules of the domain's knowledge base;
- Domain model component that describes the KBS domain knowledge, such as, domain rules;
- Ontology component that provides the terminology used in other UPML components including the Method ontology, Task ontology, and Domain ontology.
- Bridge component that establishes the relationships between two distinct UPML components. For example, the bridge between a subtask and the Domain Model sends the domain's concepts and rules to be used in resolving the subtask. This component allows the Task components and PSM component to be implemented completely detached from the Domain Model, and only receive the domain's knowledge and concepts at execution time.
- Refiner component that specializes a UPML component for a specific application.

UPML provides an approach to KBS development strongly centered on the reuse of its generic components and the use of ontologies. To resolve a knowledge intensive problem, a KBS developer can identify and reuse a specific PSM for a problem. The selected PSM must be already defined and implemented using the UPML architecture's generic components: PSM, Task, and Ontology. The developer is left with the task of defining the Domain Model and the Domain Ontology. Moreover, the growing library of generic UPML components eases the development of PSM and KBS for other tasks. For example, Pinheiro, Furtado & Furtado [12] describe a set of UPML generic components implemented in Java including the abstract-and-match PSM [16] used for KBS that solve assessment problems.

3. A UPML COMPONENT FOR KBS WEB EXPLANATIONS

3.1. General Description

In this section we introduce WebExplain, a UPML Explanation Component, responsible for generating PSM and Tasks justifications in PML. WebExplain justifications support two kinds of explanations: about the structure of the reasoning process implemented by PSMs (strategic explanations); and about the execution of subtasks, which are executions of domain's rules and concepts (domain explanations).

Figure 1 shows how WebExplain interacts with the PSM and Task components. Generally, WebExplain receives the subtask that is to be executed from the PSM component and associates all the inference steps generated from that point up to this subtask. Associations between the PML node sets and subtasks in justifications will ultimately determine the PSM's order of execution. WebExplain receives the rules that have been fired from the Task component (subtasks). Rules are provided along with their conditions and actions and recorded as inference steps of each subtask. Note that the justification generation process does not interact with the Domain Model, i.e., the inference steps are received from generic UPML components. For this reason, WebExplain is domain-independent and can be reused by other applications.

WebExplain is composed of the following classes:

- The ProofGeneration class that is used for building inference steps from parameters received from the PSM and Task components. These parameters are represented as variable bindings in the justifications. This class is responsible for encoding the current state of some of the PSM variables as sentences in the justifications. PSM sentences are written in the Knowledge Interchange Format (KIF).

[1] The terms *justification* and *proof* are used interchangeably in this paper.

- The Proof class represents a step in a justification. Inference steps are recorded from ProofGeneration and are identified by a conclusion and a set of antecedents. The conclusion of an inference step can be either derived or asserted. If the conclusion is derived, the class keeps information about the inference engine and inference rule used to derive the conclusion, e.g., JEOPS and modus ponens, and information about the premises.. If the conclusion is asserted, the class keeps information about the source, e.g. a domain ontology. To identify the inference engine generating an inference step, we have created a relationship between this class and its subclasses, i.e., UPMLProof, JEOPSProof, JESSProof, etc. A single justification generated by the ProofGeneration class can represent inference steps generated by multiple inference engines such as JEOPS [5] or JESS [7], as well as inference steps generated by the PSM control structure.
- The IWHandler class is responsible for mapping justifications, which are composed of Proof objects, into node sets and for generating PML documents.

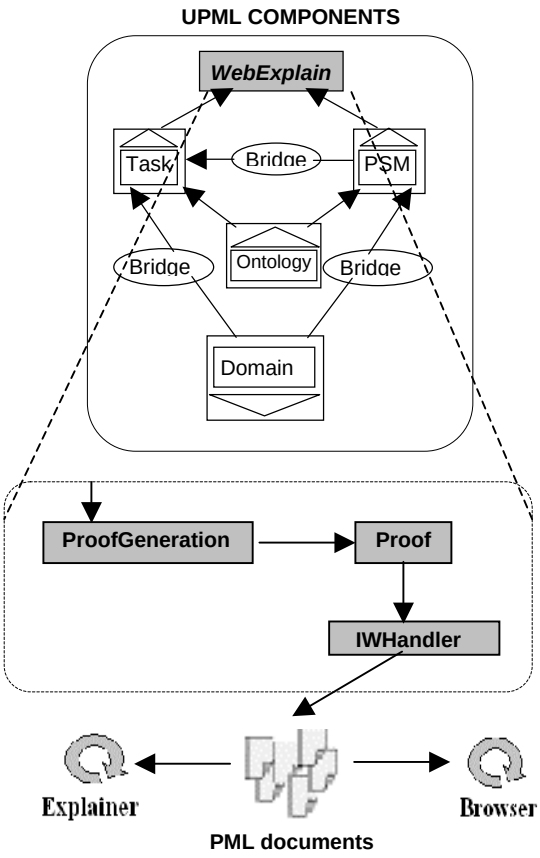

Fig. 1. The UPML original framework extended with WebExplain**.**

It is important to point out that the reasoning steps represented in PML are explored through IW services such as Explainer and Browser.

3.2. WebExplain Representation

PSM justifications can be translated into explanations since each node set in the justification is associated with the subtask wherefrom it was generated and to the node sets holding the antecedents. Moreover, the justification identifies the order in which the subtasks were executed, forming a proof tree that portrays the order defined in the PSM.

Domain explanations are generated from the inference steps associated with subtasks. For instance, a subtask can be implemented by means of the inference engine that executes rules of the domain's knowledge base or by means of an algorithm. We will call these two cases, respectively, subtask implementation A and B. Due to the generality of our approach. WebExplain should not present restrictions to KBS developers regarding the inference engines that can be used. Information referring to the conditions and actions of each rule can be retrieved from a rule ontology without the need for customizing a specific inference engine.

Figure 2 presents the classes that define the rules ontology. The Rule class defines domain rules with the name, description, rule-type, actions, and conditions slots. The actions and condition of a rule are expressions of class *Expression* defined by a name, description, expression-type, domain-variable, operator, and value slots. The domain-variable slot represents a concept of the domain manipulated by an expression. This slot is of class *Element*, which is a concept from the Domain Ontology. Instances of these classes form the domain knowledge base, whose semantic interpretation is defined by the ontology of the domain and the conditional (conditions → actions).

Tools like Protégé are widely used by the Knowledge Engineering community for creating ontologies. Moreover, tools of this category often possess plugins [14] that generate ontologies in several formats including OWL, Jess, and JEOPS. These capabilities offer two important advantages for our solution: (i) since rules are instantiated from the Rules ontology, they can be generated automatically in the format that inference engines happen to process; (ii) users of KBS or knowledge engineers can edit and evolve the domain's rules knowledge base, with no need for knowledge of inference engine languages.

The process of generation of proofs steps executed by the WebExplain is made up of the following steps:

1. It receives, from the subtask, the rule fired by inference engine (subtask implementation A). This is possible because every inference engine possesses a service, generally called Listener, that informs the sequence rules fired;
2. It executes a method of its ProofGeneration class that has access to the rules ontology to retrieve information on the conditions and action of the rule fired, as well as the classes of the domain manipulated by the rule;
3. It executes a method of its ProofGeneration class that inserts the steps of proof as objects of the Proof class based upon the conditions and actions of the rule. For example, let **p** be a wff (well-formed formula) that represents the set of the rule's conditions and let **q** be a wff that represents the set of the rule's actions. As the rule was fired, all of the rule's conditions were satisfied by facts **r,s,t**..., therefore the truth-value of **p** is true. By the rules ontology, the truth-value of (**p** → **q**) is true. Therefore, WebExplain can

insert the proof steps **q,r,s,t**..., and (**p** → **q**) as Proof objects. Proof step **q** represents a derived conclusion and is generated with information on its antecedents **r,s,t**..., e (**p** → **q**), the ModusPonens inference rule used in its deduction and the inference engine used.

4. It returns to Step 1 for the next rule fired, associating all of the proof steps to the current subtask.

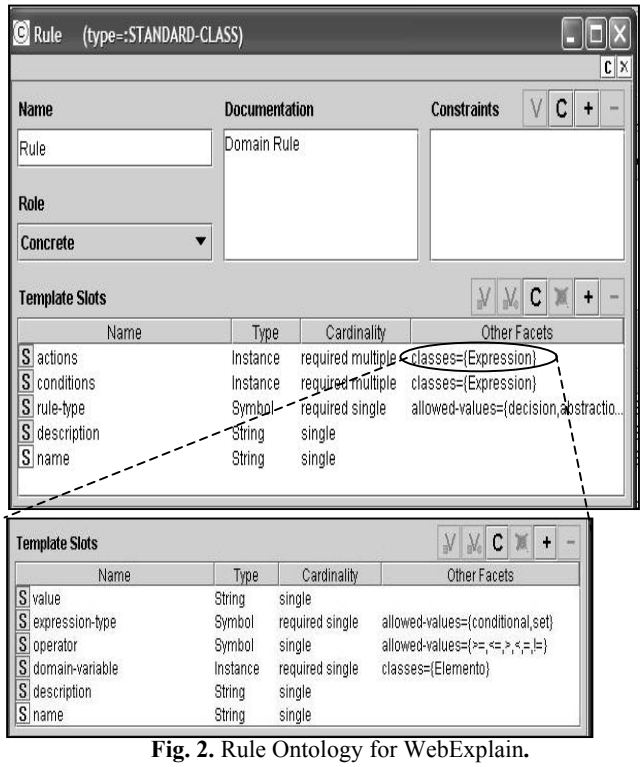

Fig. 2. Rule Ontology for WebExplain**.**

In the case where the subtask is implemented by means of an algorithm (subtask implementation B), the implementation of the algorithm must define the conditions and actions for executing each reasoning step embedded in subtask and the generation process is started from step 3, described above.

The integration between WebExplain and the generic UPML components—PSM and Task—is defined at the time of implementing the PSM, therefore the effort is performed only once, and is ready to be reused in the development of several Explainable KBS.

4. USING WEBEXPLAIN AS SUPPORT TO THE DEVELOPMENT OF EXPLAINABLE KBS

We have two kinds of reuse in our approach: the reuse of the WebExplain component in the modeling and implementation of different PSM, and the reuse of these PSM in the development of different KBS which implies, consequently, the extensibility of the WebExplain to several KBS and in its consolidation as a means of support to Explainable KBS developers.

In this section, we exemplify our approach by describing its use in the development of the abstract-and-match PSM, for assessment tasks, and in the development of an Explainable KBS—the ExpertCop System [6].

4.1. Using WebExplain in the Development of a PSM

The abstract-and-match PSM is defined conceptually in [16]. Its reasoning process defines a control structure executing the following subtasks sequentially:

1. Abstract that simplifies the case data;
2. Specify that finds criteria relevant to the case data;
3. Select that selects one criterion for evaluation;
4. Evaluate that evaluates the select criterion with respect to the case data;
5. Match that checks whether the criteria that were evaluated lead to a decision. The select, evaluate, and match subtasks are interactively executed for each criterion until a decision can be found or the criteria set is exhausted.

Basically, the AbstractMatch java class implements the PSM reasoning process through a control structure that is responsible for sequencing the subtasks. This class extends the PSMComponent class which contains generic methods to perform the mapping with the other UPML components and, among others, a method to execute the calls to subtasks—executeSubTask method. In this method, the integration with WebExplain was inserted calling a method from the ProofGeneration class, which receives the subtask that is to be executed. This integration ensures that the proof tree to be generated in PML mirrors the reasoning structure embedded in the PSM. At the end of the execution of subtasks, the AbstractMatch class invokes the publish method of the IWHandler class responsible for generating PML documents corresponding to the proof tree.

The abstract-and-match PSM subtasks are implemented as subclasses of the Java TaskComponent class and all possess a method called execute, which contains the implementation of the reasoning part destined to each one. In this method, commands must be inserted for integration with WebExplain: receive from inference engine each rule fired and call a method of the ProofGeneration class to access the rules ontology (subtask implementation A) or define the conditions and actions for executing each reasoning step (subtask implementation B); and insert the proof steps corresponding to the execution of the rule as objects of the Proof class. The Abstract, Evaluate and Match subtasks were implemented by means of an inference engine and the Specify and Select subtasks were implemented via algorithm.

The abstract-and-match PSM also define the following knowledge roles: case description, criteria to be evaluated, abstraction rules, evaluation rules, and decision rules. These knowledge roles were implemented as java classes containing generic properties and methods to receive, as parameters, the corresponding instances from classes from the domain's ontology and the rules ontology.

The development of a KBS for assessment tasks using the

UPML components in any domain can thus reuse the PSM implementation, freeing developers for implementing only domain-specific classes and for defining those domain-specific knowledge roles.

4.2. Using WebExplain for Developing an Explainable KBS

The ExpertCop System [6] is a UPML-based KBS example performing an assessment task implemented by the abstract-and-match PSM. In ExpertCop's decision-making process, a criminal cognitive agent must evaluate data gathered from a geo-simulated environment using a set of criteria to decide whether or not to commit a crime. This system is used to teach police officers about when and where crimes are likely to be committed.

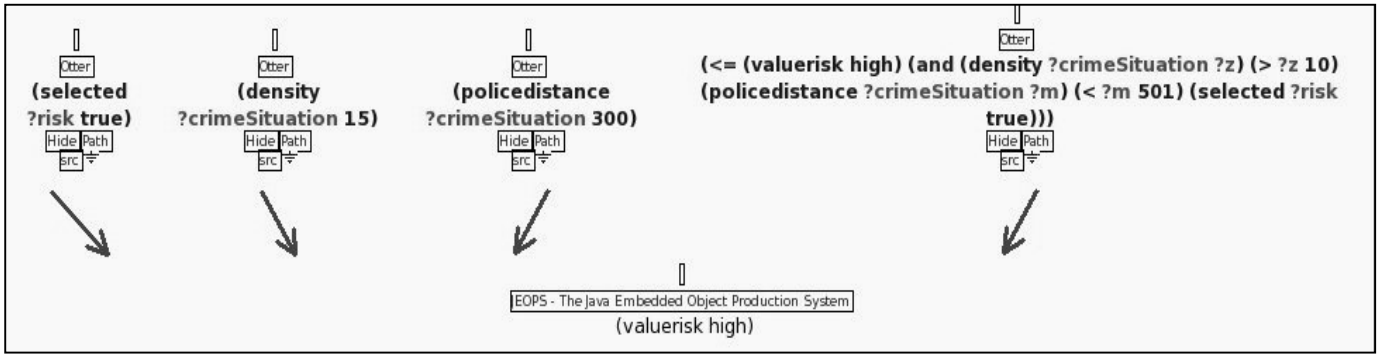

Fig. 3. Nodesets generated by the WebExplain from the subtask evaluate communication about the execution of the rule **evaluateRiskHigh**.

The developers of the ExpertCop system reused the entire implementation of the abstract-and-match PSM and its integration with WebExplain. They only had to do the domain ontology modeling and the domain rules as subclasses of knowledge roles defined by the PSM: the case description was modeled by the CrimeSituation class, the criteria to be evaluated class was modeled as CrimeCriterion, and the abstraction rules, evaluation rules, and decision rules were modeled in the rules ontology that defined rules of three types: abstraction, evaluation, and decision. These rules are processed by the JEOPS inference engine, and passed on as parameters, at execution time, to the Abstract, Evaluate, and Match subtasks.

Let us take as an example the rule evaluateRiskHigh of the evaluation type. This rule possesses the following conditions:

- CrimeSituation.density > 10;
- CrimeSituation.policeDistance < 501;
- Risk.selected = true;

and the following action:

- Risk.truthValue = high

In Figure 3, the IW browser presents a justification, in KIF, for the fact that valueRisk is high. The node sets were generated by WebExplain from the execution of the rule evaluateRiskHigh, which was chained by the subtask evaluate. The fact "valueRisk high" is encoded in a PML node set conclusion and each inference step in the node set corresponds to a fact justification. When rendering the fact justification, the browser shows that the answer, through an application of the modus ponens inference rule, was inferred from the following facts (that correspond to the rule conditions):

- (policeDistance ?crimeSituation 300)
- (density ?crimeSituation 15)
- (selected ?risk true)

and the rule itself:

- (<= (valueRisk high)
  (and (density crimeSituation ?z) (> ?z 10)
  (policeDistance ?crimeSituation ?m) (< ?m 501)
  (selected ?risk true)))

5. RELATED WORK

Explanations for KBS have appeared as a significant and independent topic of study since MYCIN [1]. NEOMYCIN [2] contributed to explanation research in KBS by using an explicit representation for problem resolution strategies and by using meta-rules in explanation planning. By using meta-rules, NEOMYCIN separated the domain ontology from MYCIN's rules. This separation allowed NEOMYCIN to be more usable as an explanation infrastructure; however it was specific to MYCIN in terms of problem solving and domain representation.

The use of Ripple Down Rules (RDR) [15] presents a new paradigm for KBS in which cases are used to explain an answer to a query. RDR provide only cases as representation for explanation which is a domain-specific representation. Moreover, RDR tools, like browsers, are specific for this knowledge representation technique. Another aspect to point out is that this approach was only used in classification tasks. Therefore, it is not trivial to extend it to other knowledge tasks such as design.

WOZ [17] is a framework for explaining component-based decision-support systems. The framework is composed of functional components that represent the reasoning process, the associated cooperative visualization

agents responsible for explanation presentation and user interaction, and application domain models such as user model, agent model, and explanation strategy. WOZ incorporates some of the major trends in software engineering including explicit models, multi-agent architectures, and visualizations. However, WOZ is not easily scalable, since a new explanation strategy must be developed for each application.

Similar to our approach is the Explanation Expert System (EES) [18] framework that provides explanations about the manners the KBS used the PSM in a certain domain. As our approach does, its Explanation Generator component provides clarifications when its explanations are not understood. The major difference is that we use a web-based infra-structure for explanation which could be useful in the development of problem solvers in general (web services, agents, etc.). Moreover, this infra-structure benefits of portability. It can be shared with other applications and opens the possibility to cooperative explanations.

6. CONCLUSION

In this paper, we address the issue that KBS need explanation components and that we are not aware of any that provide a systematic way to generate explanations of PSM implementations or that can be easily reused for the development of other applications, thus reducing the implementation effort of the explainable KBS. So, we propose an extension of the UPML framework by providing a reusable explanation component—WebExplain. WebExplain is integrated to the UPML generic components: PSM and Task. Hence, we are leveraging UPML facilities as reuse and ontologies. WebExplain provides proofs from the reasoning process embedded in the PSM and subtasks, which serve as a basis for explanations about the reasoning structure and explanations about the domain's knowledge. Another characteristic of WebExplain is the use of PML for dumping proofs, therefore enabling proof and explanation interoperability between distributed applications.

We exemplify our approach in the development of a PSM: how should the integration between WebExplain and generic UPML components be implemented. We identified three points of easy integration. This effort was performed only once and the PSM, integrated to WebExplain, can be reused, freeing Explainable KBS developers from the worry of having to implement only domain-specific classes.

Future works aim at projecting some pragmatic principles of linguistic interactions onto the semantic structures of the PML proofs in order to select the information to be conveyed to users, to simplify proof steps and to reorganize proofs, in order to improve the quality of the explanations.

REFERENCES

[1] B. Buchanan and E. Shortliffe. Rule based expert systems: The MYCIN experiments of the Stanford Heuristic Programming Project, Addison-Wesley, Reading, MA, 1984.

[2] W. Clancey. From GUIDON to NEOMYCIN and HERACLES in Twenty Short Lessons: ORN Final Report 1979-1985, AI Magazine, 7(3), pp. 40-60, 1986.

[3] D. Fensel and V.R. Benjamins. Key Issues for Automated Problem-Solving Methods Reuse. 13th European Conference on Artificial Intelligence, ECAI98, Wiley & Sons Pub., 1998.

[4] D. Fensel et al. The Unified Problem-Solving Method Development Language UPML. Knowledge and Information Systems, An International Journal, 5, 83-127, 2003.

[5] C. Figueira Filho and G. Ramalho. Jeops – The Java Embedded Object Production System. IBERAMIA-SBIA 2000. LNAI 1952, Berlin: Springer-Verlag, 2000.

[6] V. Furtado and E. Vasconcelos. A Multi-Agent System to Teach Police Allocation. Proc. of 17th Innovative Application of Artificial Intelligence (IAAI-2005), Pittsburgh, 2005.

[7] JESS. http://herzberg.ca.sandia.gov/jess, as available on March 10th, 2006.

[8] D.L. McGuinness and P. Pinheiro da Silva. Infrastructure for Web Explanations. In Proceedings of 2nd International Semantic Web Conference (ISWC2003), D. Fensel, K. Sycara and J. Mylopoulos (Eds.), LNCS 2870, Sanibel Is., FL, USA. Springer, pages 113-129, October 2003.

[9] D.L. McGuinness and P. Pinheiro da Silva. Explaining Answers from the Semantic Web: The Inference Web Approach. Journal of Web Semantics. Vol.1 No.4., pages 397-413, October 2004.

[10] D.L. McGuinness and F. van Harmelen. OWL Web Ontology Language Overview. W3C Recommendation, February, 2004.

[11] P. Pinheiro da Silva, D.L. McGuinness and R. Fikes. A Proof Markup Language for Semantic Web Services. Information Systems, Volume 31, Issues 4-5, June-July 2006, Pages 381-395. Also, Stanford KSL Technical Report KSL-04-01.

[12] V. Pinheiro, E. Furtado and V. Furtado. A Unified Architecture to Develop Interactive Knowledge Based Systems. In proceedings of the 17th Brazilian Symposium of Artificial Intelligence (SBIA 2004), Bazzan, Ana L.C. e Labidi, S. (Eds), LNAI 3171, São Luís, MA, Brazil, Springer-Verlag, pp 174-183, 2004.

[13] Protégé: http://protege.stanford.edu, as available on March 10th, 2006.

[14] Protégé Plug-ins: http://protege.stanford.edu/download/plugins.html, as available on March 10th, 2006.

[15] D. Richards. User-Centred and Driven Knowledge-Based Systems for Clinical Support Using Ripple Down Rules. Proceedings of the 33rd Hawaii International Conference on System Sciences, 2000.

[16] G. Schreiber, H. Akkermans, A. Anjewierden, R. Hoog, N. Shadbolt, W. van de Velde and B. Wielinga. Knowledge Engineering and Management: The CommonKADS Methodology. The MIT Press. Cambridge, MA, 2000.

[17] R.D. Shankar, S.W. TU, M.A. Musen. A Declarative Explanation Framework That Uses A Collection Of Visualization Agents. Stanford Medical Institute, Stanford University School of Medicine, Stanford, CA, 1998.

[18] W. Swartout, C. Paris and J. Moore. Explanations in Knowledge Systems: Design for Explainable Expert Systems. IEEE Expert: Intelligent Systems and Their Applications, vol. 06, no. 3, pp. 58-64, Jun., 1991.