Extending the Unified Problem-solving Method Development Language

Calin Sandru[*] and Viorel Negru[**]

*WebQuote.com SRL Timisoara, C. Brediceanu, 8, Timisoara, Romania, csandru@webquote.com
**West University of Timisoara, Bd. V. Parvan 2, Timisoara, Romania, vnegru@info.uvt.ro

Abstract: The task concept is a widely used pattern in reasoning systems. Unified Problem-solving Method Development Language (UPML) is a framework allowing building libraries of generic problem solving components. This paper proposes several improvements in this framework based on observations when modeling a concrete application. It starts from describing in UPML an application intended to solve nonlinear equation systems and identifies places where extensions in the language are of real help.

Keywords: task reasoning, UPML, ontology, modeling, numerical analysis

1. Introduction

The AI community is working for a long time in the area of knowledge representation and their reutilization. The two main directions in the history of this domain are the *functional approach* and the *conceptual approach*.

Functionally manipulating knowledge is based on the observation that tools should be adapted to the knowledge structure instead of modeling knowledge based on the available tools. The main exponent of this approach is Chandrasekaran. He also stressed the fact that there exists control knowledge with similar structure for various domains of applications like medicine and technique and came up with the notion of *generic tasks* [3]. A generic task corresponds to a common problem and the manner it is solved. Its generic character is based on the possibility to use that task to solve complex problems in various domains. Sample generic tasks are: hierarchical classification, hypothesis matching, abductive assembling, hierarchical design by selection and refining [3]. Researches in this area include construction of various languages for describing generic tasks: CSRL [2], DSPL [1], HYPER [10], PEIRCE [17].

The next step was the notion of *task structure* that introduces a clear separation between *tasks* and the *methods* to perform the tasks. Since a method may refer other tasks (sub-tasks) in its body, a task structure is better viewed as a tree of tasks, methods and sub-tasks which are recursively applied until the elementary tasks are finally applied on available knowledge [4].

Task structures represent both a technique to analyze systems and a way to implement practical systems using specific architectures. It is worth pointing out some researches in this area like TIPS [16] or SOAR task extension [11].

If the functional modeling tries to identify tasks and the methods to realize them, the conceptual approach focus on identifying different conceptual levels in order to model the knowledge. The most important methodology developed in the area is the KADS architecture, fully described in [19]. In KADS one identifies several models used to represent the knowledge. The most relevant for the purpose of this paper is the *model of expertise* which is further divided in several layers: domain, inference, tasks. The language to describe these layers is $(ML)^2$ [9]. This language is heavily based on logic (extensions of the first order logic or modal logic), thus it requires some kind of logical expertise from the user to be used.

The Unified Problem-solving Method Development Language (UPML) described in [6] is based on the researches on KADS and KARL [12]. It aims to preserve the KADS structure of the model of expertise and also to simplify the description language as KARL does.

The remaining of this paper focuses on modeling the NESS (Non-linear Equations System Solver) application [14] in terms of the UPML language. It starts by shortly presenting the components of the UPML followed by the description of NESS in terms of these components. This process allows for the identification of three possible extensions of the UPML facilitating the NESS description.

2. Extending the UPML

2.1 UPML Description

One of the starting points of UPML consists in the identification of the essential components involved in the development of a knowledge reasoning system. There are six components that represent the basic terms for the UPML language:

- *Task:* it defines the problem to be solved. The task attributes include a textual description of the task goal, a list of ontologies providing the task terminology, a logical statement describing the task goal, a set of terms representing the task input, a set of terms representing the task output, a set of statements about dynamic input restricting the valid inputs of the task (pre-conditions) and another set of statements about task knowledge intended to show the conditions when the task goal could be reached (assumptions).

- *Problem Solving Method (PSM):* it defines the reasoning process in order to satisfy a task. The components of PSMs are a textual description, a set of ontologies used for describing it, pre-conditions on input, post-conditions on output which must hold after PSM's execution. The input and the output are other attributes, together with a list of the sub-tasks called by this method. Complex methods also require the description of the control which details the way the sub-tasks are called by this method. A bridge (see below) associates a PSM with a task, thus a task may be fulfilled by several PSMs.
- *Domain Model:* allows for the description of the domain knowledge specific to the problem. Apart from tasks and PSMs descriptions, this knowledge is not intended to be reusable.
- *Ontology:* offers the terminology used to describe task, methods and the domain model. The relations between the UPML components were accurately modeled in environments like PROTÉGÉ II where the central point is the description of ontologies [5, 8]. Ontology sharing becomes a central issue in many areas like semantic web or multi-agent technology and UPML is a promising language for modeling shared reasoning knowledge manipulated as tasks or PSMs [15].
- *Bridge:* is an adapter [7] allowing relating architectural parts like tasks with the PSMs, tasks with the domain model or PSMs with the domain model. The purpose of this component is to map terms from different ontologies (the ones used to describe the related architectural parts).
- *Refiner:* is another kind of adapter allowing refining architectural elements. It is thus possible to refine, for example, a general search method into a Greedy method by replacing some sub-tasks called in the original search method with adapted sub-tasks suited for the Greedy method (e.g. choosing of the next state).

The association between tasks and PSMs is a one-to-many relation. There is a natural issue to be solved when solving a task: how to choose the most appropriate PSM from a list of candidates. UPML introduces the *competence* part in PSM's description comprising the input and output roles specification together with pre-conditions and post-conditions restricting the valid input and output. Discussion on task-PSM relation allowed us to propose two of the UPML extensions in this paper.

2.2 NESS Description in UPML

This section is intended to shortly describe the NESS (Nonlinear Equations Systems Solver) system [14] and to show how UPML could be used to model this application.

In general, there is not a unique recipe in solving the non-linear equations systems (non-linear simultaneous equations) [13]. To choose a particular method one should take into account several external aspects: the form of the system, knowledge about a "good" iteration, structural characteristics (the quality of the Jacobian, condition number), etc.

Sample solving methods for non-linear equations systems (NES) considered in NESS are the Classic Newton method, the Steffensen method, the Conjugated Gradient method, or Broyden methods. Any of them has particular properties like the convergence rate or precision.

The overall goal of NESS is to develop an intelligent system able to assist the user in solving non-linear equations systems. Modeling the human expertise, the system must be able to automatically choose the most adequate numerical methods or to combine two or more numerical methods for solving non-linear equations systems. The system assumes coupling the symbolic and numeric calculus.

NESS has the following more concrete goals:

- the acquisition and the formalization of the expert knowledge like solving methods' properties and their mapping on problem properties
- the identification of problem properties of interest in further reasoning
- the development of the solving methods base and of the corresponding routines library
- the design of a task oriented formalism to model the problem solving reasoning

The figure above shows the hierarchy of tasks and PSMs modeling the NESS application together with the ontologies defining the problem terminology.

From the figure it can be seen that several ontologies are used in order to define the problem terms. The figure shows the import relations between these ontologies. For example the ontology *Equation Systems* is imported in *Nonlinear Equation Systems* which is further imported in a more specific ontology called *Numerical Solving of Nonlinear Equations*. Another important ontology is one that defines the terms for iteration: *Solving by Iteration* which inherits from *Approximation*. The domain model is called *Nonlinear System Solving* and is related by bridges with several tasks like *Iteration* or *Approximately Solving NES*. Tasks are related by bridges with PSMs. For example, the *Iteration* task has two PSMs possible to realize it: *Classic Newton*or the *Combined* method. You may also note that there are PSMs like *NewtonIteration* that can implement two tasks: *Iteration Generation Newton* and *Iteration Generation Combined*.

One may also note that the *IterativeSolving* PSM is refined to PSMs like *Classic Newton* or *Combined Method*.

There are two relations in the figure which do not pertain to the original UPML description. They represent enhancements proposed in this paper and they will be analyzed below.

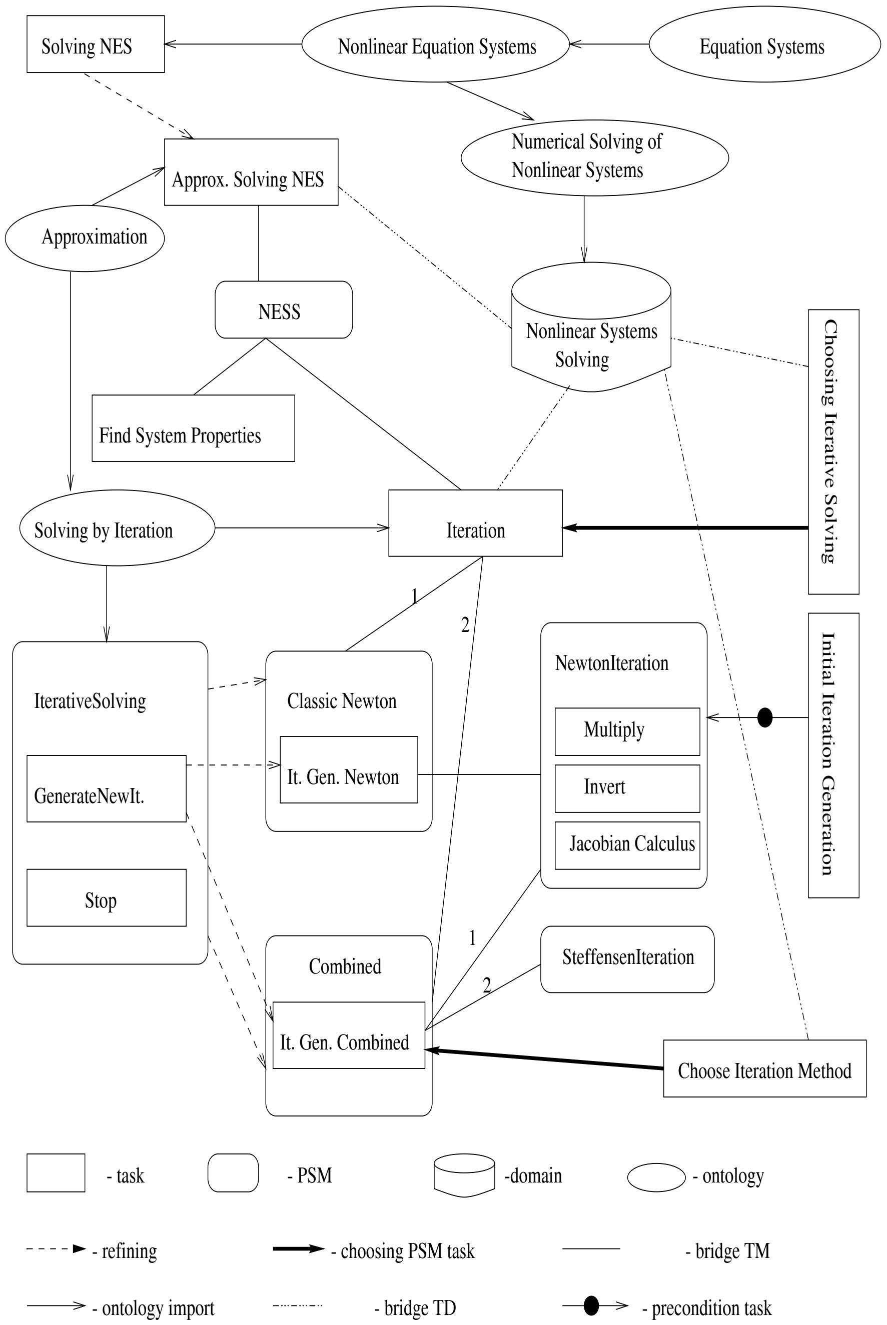

Figure 1. The NESS hierarchy of tasks, methods and ontologies.

2.2.1 Choosing the PSM to Fulfill a Task

The first new relation refers to the way the task *Choosing Iterative Solving* is linked with the *Iteration* task by an element of the type *choosing PSM* (bold arrow). As illustrated in the section 2.1, the UPML language includes the PSM competence description that, together with the ontology bridges, facilitates the choosing of the appropriate PSM to perform a task. An agent could choose the right method to apply based, for example, on the checks it can perform on the actual task input using the PSM precondition. It also may check the availability of bridges between various ontologies used to describe the PSM's actual input. It results a choosing process based only on *ontological* knowledge. It means that a PSM will not be considered as the method to fulfill a task if, for example, the class in the ontology describing the method input in UPML description could not be bridged with a class in the ontology describing the task input for the actual problem domain. It also means that if the post-conditions of the method (described in UPML) assume, after the method execution, the existence of some objects described in the method ontology whose classes could not be matched with classes in the ontology describing the problem domain, the PSM is not considered for this task.

In the case of the NESS problem, one can see in the figure that there are the *Classic Newton* method and the *Combined* method as candidates for the *Iteration* task. There are some more solving methods since there are many mathematical methods to iteratively solve a NES. Apart from the type of filtering described above, NESS is willing to make a complex choice based on methods properties available as domain knowledge (not as PSMs preconditions). NESS is planning to use an expert system for making this decision since this is a complex task and deserves a special treatment. It is exactly the purpose of the **`method-chooser-task`** specification item in PSM's description to assign a task for choosing the PSM to solve the *Iteration* task. Part of the UPML definition for the *Iteration* task is presented below:

```
task Iteration
  pragmatics
   General iteration task
  ontology
    Solving by Iteration
  specification
    roles
      input function; input initialIteration; input
error; input steps;
      output iterations;
    method-chooser-task
      ChoosingIterativeSolving(function,
initialIterations)
    goal
      task(input function; input initialIterations;
            input error; input steps;
```

```
          output iterations) →
                    newIterations(iterations,
    function, initialIterations, error, steps) ∧
        (initialIterations ⊂ iterations);
  preconditions
        steps > 0;
```

The *ChoosingIterativeSolving* task must reason about PSMs based on the domain knowledge. As a consequence, it must import the UPML ontology and the developer of the system must create bridges between the domain model and the UPML ontology. The input of this task is intended to be the actual input of the task it is linked with. The output of the task is an object of class PSM according to the UPML ontology.

2.2.2 PSM's Precondition Satisfaction

The second relation introduced in this paper is reflected in the link between the task *Initial Iteration Generation* and the PSM *NewtonIteration* (arrow with bullet). The task in the relation is called *precondition task* because its role is to satisfy a PSM's precondition if that precondition is false. The reason to add this new construct for our problem is based on the necessity to provide initial iteration(s) for a NES PSM. A second factor affecting this decision is related to the fact that the *NewtonIteration* PSM may be or may be not called with the initial iterations already specified. For example, the task *Iteration Generation Combined* may be satisfied using the *NewtonIteration* PSM, but since it is called in a while loop in the *CombinedMethod* the initial iteration already exists for iterations after the first call of *Iteration Generation Combined*. The code extract showing the new notation is presented below in the preconditions section:

```
problem solving method NewtonIteration
  pragmatics
    Iteration generation based on a Newton process
  ontology
    Iterative Solving
  competence
    roles
      input fct; input iterations; output newIteration;
    preconditions
    iterations <> Ø |
      iterations := InitialIterationGeneration(fct,
'ClassicNewton');
    subtasks
      Multiply, Invert, JacobianCalculus;
    postconditions
      …
```

2.2.3 Tasks with Pre/Post-tasks

There is a third enhancement proposed in this paper, whose purpose from NESS is not reflected in the figure. However, there is a case where it may be used as illustrated below.

In some situations, an alternative approach to the adding of the precondition satisfaction task (see above) is to assure that the *Initial Iteration Generation* task is executed before the task intended to be realized by the *NewtonIteration* PSM. In the NESS application, an appropriate choice would be to mark this task as a *pre-task* of the *Iteration* task. Symmetric considerations allow for introducing *post-tasks* for a specified task. Thus, this paper proposes explicitly adding the notions of post/pre tasks in UPML task structure as a new UPML relation, like exemplified below:

```
task Iteration
  pragmatics
    General iteration task
  ontology
    Solving by Iteration
  specification
    roles
      input function; input initialIteration; input
error; input steps;
      output iterations;
    method-chooser-task
      ChoosingIterativeSolving(function,
initialIterations)
    pre-tasks
      InitialIterationGeneration
    post-tasks
      PostIterationTask1;PostIterationTask2
    goal
      …
    preconditions
      …
```

The usefulness of the notions of pre/post tasks is also proved by some other task based environments like SCARP [20]. The paper [18] uses this relationship in a context where several agents perform tasks and there were restrictions on the task ordering.

Conclusions

As presented in the previous section, three new elements were proved to be useful to be included in UPML for the benefits of the NESS application: a task for choosing PSMs for another task, precondition satisfying task and the pre/post tasks. The first allows for enhancing the process of choosing a PSM to satisfy a task by using domain expertise. It is very valuable in this mathematical context since there is complex domain knowledge to be used in the choosing process apart from the ontological matching. The second allows postponing the elimination of a PSM in the process of selection of the method to fulfill a task by specifying an alternative way to fulfill a PSM's precondition and the third new proposed element represents a way to make explicit an order relation between tasks.

The present work was prepared in order to model the task reasoning in multi-agent systems. Of interest in such contexts would be to model the execution of a task in agents' local data environment described using some appropriate domain model. This remains as a future work for us to propose UPML extension with appropriate constructs.

References

[1] D.C. Brown, B. Chandrasekaran. Expert Systems for a Class of Mechanical Design Activity. Knowledge Engineering in Computer-Aided Design, (ed. J.S. Gero), pp. 259-282, North-Holland, New York, 1985.

[2] T. Bylander and S. Mittal. A language for classificatory problem solving. AI Magazine, 7(3):66-77, 1986.

[3] B. Chandrasekaran. Generic Tasks in Knowlwdge-Based Reasoning: High-Level Building Blocks for Expert System Design. IEEE Expert, pp. 23-30, Fall 1986.

[4] B. Chandrasekaran. Design Problem Solving: A Task Analysis. AI Magazine, 11(4):59-71,Winter 1990.

[5] M. Crubezy, Z. S. Pincus, M. A. Musen. Mediating Knowledge between Application Components. Semantic Integration Workshop of the Second International Semantic Web Conference (ISWC-03), Sanibel Island, Florida, CEUR, 82. 2003.

[6] D.Fensel, E. Motta, V.R. Benjamins, M. Crubezy, S. Decker, M. Gaspari, R. Groenboom, W. Grosso, F. van Harmelen, M. Musen, E. Plaza, G. Schreiber, R. Studer and B. Wielinga. The Unified Problem-solving Method Development Language UPML. Knowledge and Information Systems, 5(1), 2003.

[7] E. Gamma, R. Helm, R. Johnson and J. Vlissides. Design Patterns. Elements of Reusable Object-Oriented Software. Addison-Wesley, 1995.

[8] J. H. Gennari, M.A. Musen, R.W. Fergerson, W.E. Grosso, M. Crubézy, H. Eriksson, N.F. Noy, S.W. Tu: The evolution of Protégé: an environment for knowledge-based systems development. Int. J. Hum.-Comput. Stud. 58(1): 89-123, 2003.

[9] F. van Harmelen and J. R. Balder. $(ML)^2$: a formal language for KADS models of expertise. Knowledge Acquisition Journal. 4(1):127-161, 1992.

[10] T.R. Johnson, J.W. Smith and T. Baylander. HYPER - Hypothesis matching using compiled knowledge. Proceedings of the AAMSI Congress 1989

American Association for medical Systems and Informatics, San Francisco, California, pp. 126-130, 1989.

[11] K.A. Johnson, T.R. Johnson, J.W. Smith, M. DeJongh, O. Fisher, N.K. Amra and A. Bayazitoglu. RedSoar: A System for Red Blood Cell Antibody Identification. Proceedings of the Fifteenth Annual Symposium on Computer Applications in Medical Care, pp. 664-668, McGraw Hill, Washington D.C., 1991.

[12] D. Landes. DesignKARL - a Language for the Design of Knowledge-Based Systems. 8th Knowledge Aquisition for Knowledge-Based Systems Workshop KAW94, Banff, Canada, 1994.

[13] S. Maruster. Numerical methods in nonlinear equations solving. Editura Tehnica, Bucuresti, 1981.

[14] V. Negru, St. Maruster, C. Sandru: Intelligent system for non-linear simultaneous equation solving, Tehnical report, RISC-Linz Report Series, No. 98-19, December, 1998.

[15] B.Omelayenko, M.Crubézy, D. Fensel, V.R. Benjamins, B.J. Wielinga, E. Motta, M.A. Musen andY. Ding: UPML: The Language and Tool Support for Making the Semantic Web Alive. Spinning the Semantic Web 2003: pp. 141-170, 2003.

[16] W.F. Punch. A Diagnosis System Using a Task Integrated Problem Solving Architecture(TIPS), Including Causal Reasoning. PhD. Thesis, Dept. of Computer and Information Science, The Ohio State University, 1989.

[17] W.F. Punch, M.C. Tanner, J.R. Josephson and J.W. Smith. PEIRCE: Atool for experimenting with abduction. IEEE Expert, 5(5):34-44,1990.

[18] C. Sandru, V. Negru and D. Pop - A Multi-Agent Approach to a Sales Optimization Application, The 14th International Conference on Control Systems and Computer Science July 2 - July 5, 2003, Bucharest, Romania.

[19] G. Schreiber, H. Akkermans, A. Anjewierden, R. de Hoog, N. Shadbolt, W. Van de Velde and B. Wielinga. Knowledge Engineering and Management. The CommonKADS Methodology. The MIT Press, 1999.

[20] J. Willamowski. Modélisation de tâches pour la résolution de problèmes en coopération système-utilisateur. Actes 9ième RFIA, Paris (FR), pp305-316, (11-14 janvier) 1994.