Developing ontological model of computational linear algebra – preliminary considerations

Katarzyna Wasielewska*, Maria Ganzha*, Marcin Paprzycki* and Ivan Lirkov,†

*System Research Institute Polish Academy of Sciences, Warsaw, Poland
†Institute of Information and Communication Technologies, Bulgarian Academy of Sciences, Sofia, Bulgaria

Abstract. The aim of this paper is to propose a method for application of ontologically represented domain knowledge to support Grid users. The work is presented in the context provided by the *Agents in Grid* system, which aims at development of an agent-semantic infrastructure for efficient resource management in the Grid. Decision support within the system should provide functionality beyond the existing Grid middleware, specifically, help the user to choose optimal algorithm and/or resource to solve a problem from a given domain. The system assists the user in at least two situations. First, for users without in-depth knowledge about the domain, it should help them to select the method and the resource that (together) would best fit the problem to be solved (and match the available resources). Second, if the user explicitly indicates the method and the resource configuration, it should "verify" if her choice is consistent with the expert recommendations (encapsulated in the knowledge base). Furthermore, one of the goals is to simplify the use of the selected resource to execute the job; i.e. provide a user-friendly method of submitting jobs, without required technical knowledge about the Grid middleware. To achieve the mentioned goals, an adaptable method of expert knowledge representation for the decision support system has to be implemented. The selected approach is to utilize ontologies and semantic data processing, supported by multicriterial decision making. As a starting point, an area of computational linear algebra was selected to be modeled, however, the paper presents a general approach that shall be easily extendable to other domains.

Keywords: semantic data processing, domain knowledge, user support
PACS: <Replace this text with PACS numbers; choose from this list: `http://www.aip..org/pacs/index.html`>

INTRODUCTION

The aim of this paper is to discuss practical aspects of application of ontologies and semantic data processing to support users of Grid middleware. The context is provided by the *Agents in Grid* (*AiG*; [1, 2, 3]) system, which aims at the development of an intelligent agent-based meta-level middleware for the Grid. In the *AiG* system, software agents represent Grid users (both resource consumers and resource providers). In this paper we focus our attention on the following scenario. Let us assume that the user would like to submit a job to be executed in the Grid. As discussed in [4], in a real world settings (exemplified by the computing infrastructure of the University of Huddersfield), (i) users are not willing to spend time to learn how to use Grid middleware, and (ii) once they learn how to use a specific method / library / code to solve their problems, they are not inclined to change their habits. Specifically, they tend to solve the problem with the same method using the same resource configuration, which they mastered, despite the fact that there may be more suitable algorithm and / or computational resource available in "their Grid." As mentioned in [4], the Grid middleware of choice in many scientific institutions e.g. University of Huddersfield (UoH) is the Globus Toolkit (GT; [5]), which is mature and efficient. However, it lacks features such as "resource discovery for users." This means that a serious burden is placed upon system administrators, to continually publish up-to-date resource information so that the users may be able to submit jobs to the evolving systems. At the UoH, the decision was made to additionally use the European Middleware Initiative (EMI; [6]) middleware that enables the job submission system to communicate directly with the information gathering systems; allowing up to date resource information to be collected, and then allows users to submit their jobs using a single scripting language. However, even this configuration still requires knowledge of technical details of the Grid (e.g. the submission scripting language), and the interplay between the problem to be solved, the algorithms, the libraries that implement them, and the available computer hardware.

Therefore, for the solution to be regarded as fully user-friendly, the job submission system (to the "Grid system") should support the user in defining the task / problem to be executed / solved, help her with the selection of the optimal method / algorithm / tool to solve it, as well as with defining the criteria to select the needed resource(s). Then, such system should simplify the use of the selected resource(s) (from here on, the term "resource" may mean either hardware or software resource; depending on the context). Specifically, the job submission process should be the

same regardless of the Grid middleware installed on a given resource, and no knowledge of scripting language should be required. In this paper we focus on the first aspect of the problem, helping the user to select the right resources. Note that, the proposed expert system is dedicated most of all to Grid users without in-depth knowledge about relation between problems, algorithms and respective technical requirements. However, it can be also utilized by users who want to verify if their approach to solve a given problem is consistent with available expert recommendations. Let us start from a brief analysis of the related work.

Related work

It is easy to observe that our work represents another attempt at achieving goals summarized in [7, 8]. Michael Lucks's dissertation focuses on the problem of software selection for solving mathematical problems. In his work, a generic mechanism for representing the software selection related expertise was proposed, i.e. domain experts could express functional relationships between properties of mathematical problems and the performance of the software. In the proposed representation scheme, attention was paid to evaluate how suitable is a given software for a given problem. The suitability of a software was calculated with a set of functions expressing knowledge about the effects of the problem properties on the performance. Moreover, author discusses the problem of knowledge acquisition. In the case of the *AiG* system, the assessment of suitability is to be handled by experts that construct the knowledge base. Therefore, the mechanisms described by M. Lucks can be utilized to improve the quality of their recommendation(s). The latter issue corresponds to the complex and time-consuming task that we also try to address; i.e. how to construct a knowledge base that can be easily extended and handled. In [7] the process of acquisition of new knowledge required assistance from a "knowledge engineer," due to the complexity of the functional relationships. Note that, another aspect of designing a knowledge representation scheme is it's transparency and ease of use. Here, we hope that with growing popularity of ontologies and semantic data processing, new generation of tools will be developed to help solving this problem. We believe that expressivness of ontological languages allows to specify expert knowledge in an intuitive and user-friendly way.

Practical aspects of software selection were discussed in [8]. There, the authors propose a numerical problem solving environment (named *EPODE*), which supported solving initial value problems for ordinary differential equations. The expert features of the system include (i) specifying the problem, (ii) selecting algorithmic alternatives, (iii) determining problem-oriented parameters of the algorithm, and (iv) identifying problem properties that can indicate which method is suitable. Even though the *EPODE* was (and still is) an advanced tool that provides a broad functionality, it was dedicated only to ordinary differential equations and, even more specifically, to initial value problems.

Other expert systems in the area of differential equations that deal with software selection problem are described in [9, 10, 11]. They suggest the best numerical solver (for partial differential equations) based on input data properties. Furthermore, additionally they can generate an appropriate execution code. The decision mechanisms are based, for instance, on decision trees or data mining of the performance history. Here, let us note that the support mechanism that is planned for the *AiG* system, is going to use a decision mechanism that does not use hard-coded decision rules characteristic for a given domain, but a multicriteria decision method to select optimal expert recommendation, based on expert knowledge that is stored in the *AiG* system knowledge base. Moreover, the knowledge representation scheme (common conceptual model) in the *AiG* can be applied to different domains due to its adaptable nature.

Unfortunately, the presented work does not seem to attract much attention. According to our consultations with Grid computation specialists, research centres rarely (if never) use expert systems to assist users with Grid utilization. Usually, the situation is analogous to the one at UoH, where users have to master technical details and have knowledge about correlations between problem properties and respective solution methods. We hope that with application of modern tools, such as ontological representation of domain knowledge and semantic data processing, we will be able to obtain some insightful effects in the area of application of domain knowledge to support Grid users.

DOMAIN KNOWLEDGE ONTOLOGY

The key challenge in designing and implementing any user decision support system is the construction of the knowledge base. In our case it is going to be an ontology that stores expert knowledge regarding selected domain(s). Note that, *all* knowledge in the *AiG* system is stored in / represented as an ontology. Furthermore, communication protocols utilize messages with ontological content (for details see [12]). Specifically, during the development of the

system, a set of ontologies was designed to provide concepts necessary for: (i) resource and Grid structure description (*AiG Grid Ontology*), (ii) contract requirements specification (*AiG Conditions Ontology*), and (iii) content of messages exchanged in the system (*AiG Messaging Ontology*). Obviously, to develop the user support functionality, described in previous sections, a need arise to add a new *AiG Expert Ontology*. The position of the new ontology in the hierarchy of ontologies used in the *AiG* system is shown in Figure 1.

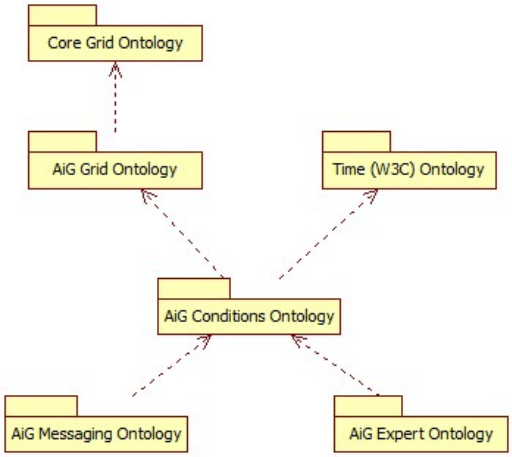

FIGURE 1. Hierarchy of *AiG* ontologies

The ontology with expert knowledge is an application ontology, which imports the *AiG Conditions Ontology*. Furthermore, it uses the concept of the *Grid resource* from the *AiG Grid Ontology*, to point to resource(s) that is / are most suitable for solving a given problem (according to the knowledge stored in the knowledge base).

As a starting point, and as a proof of concept for the selected approach to implement user support, we focus on the domain of computational linear algebra. The ontology for this domain is described in detail in the following subsections. However, it should be stressed that the structure that we design should be applicable to any domain (to be later used within the system). Here, recall that jobs that can be executed in the Grid, span multitude of domains.

For the task at hand, formalized concepts have to capture three main aspects of a domain (this is the above mentioned structure of the ontology that can be applied to other domains):

- **Problem** – hierarchy of problems from a given domain.
- **Algorithm** – algorithms / methods that can be used to solve problems from a given domain (problems specified in the *Problem* part of the ontology).
- **Data Element** – type of input data (objects) that algorithms / methods operate on. We use this concept to indicate the type of the object: (i) simple e.g. boolean, character, integer, real, or (ii) structured e.g. matrix, string and assign properties e.g. for structured data elements: *hasDimension*, *hasBlockDimension*, *hasSimpleType*.

Additional concept that is used to describe a job is the **Data Property** – the hierarchy of properties that characterize the input data. For the computational linear algebra, properties are grouped into two subclasses: *Matrix Property*, and *Matrix Element Property*. The *Data Property* class is a range for the *hasProperty* object property that relates the instance of the *Data Element* class and its characteristics. Other concepts from the ontology, used to match the expert recommendations, are (see Figure 2):

- **Domain Expert** – concept representing experts (people or computer system) that provide recommendations within a given domain. Property *hasExpertOpinion* points to instances of the *Expert Opinion* that were provided by a given expert.
- **Job Profile** – concept that relates instances of: *Problem*, *Data Element* and *Algorithm* classes, with instances of the *Expert Opinion* class. The respective object properties to be used are: *forData*, *forProblem*, *hasAlgorithm*.
- **Expert Opinion** – concept that relates instances of the *Domain Expert* and the *Grid Entity* classes, i.e. the resource recommended for solving a specific problem indicated with the instance of the *Job Profile* class. Obviously,

resources originate from the *AiG Grid Ontology*. Here, the applicable properties are: *hasRecommendedResource proposedByExpert*, and *hasJobProfile*.

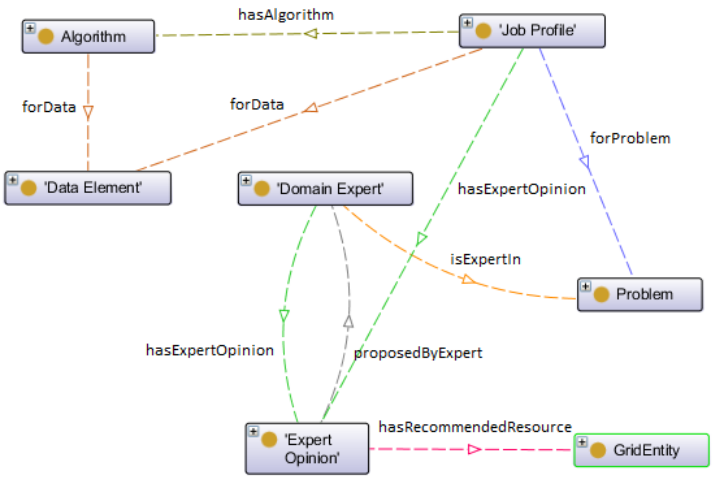

FIGURE 2. Main concepts in *AiG Expert Ontology* involved in matching expert recommendations

The expert ontology is to provide concepts necessary to capture main aspects of a given domain and at the same time retain structure that is universal for any domain. It can be noticed that a new domain knowledge can be added to the ontology by extending the hierarchy of subclasses for any of the main concepts e.g. *Problem*, *Algorithm* and *Data Property* classes. Alternatively, ontologies for different domains can be stored in separate OWL ontology files, but the top level concepts and properties will be common.

Hierarchy of problems

Let us now present the hierarchy of problems distinguished in the area of computational linear algebra (see Figure 3). The respective OWL classes represent the following problems: (i) eigen problem, (ii) least squares problem, (iii) solution of a system of linear equations, (iv) singular value decomposition, and (v) calculation of a matrix norm. Each of these problems can be further decomposed. Based on consultations with several experts, we believe that this conceptualization is fairly complete.

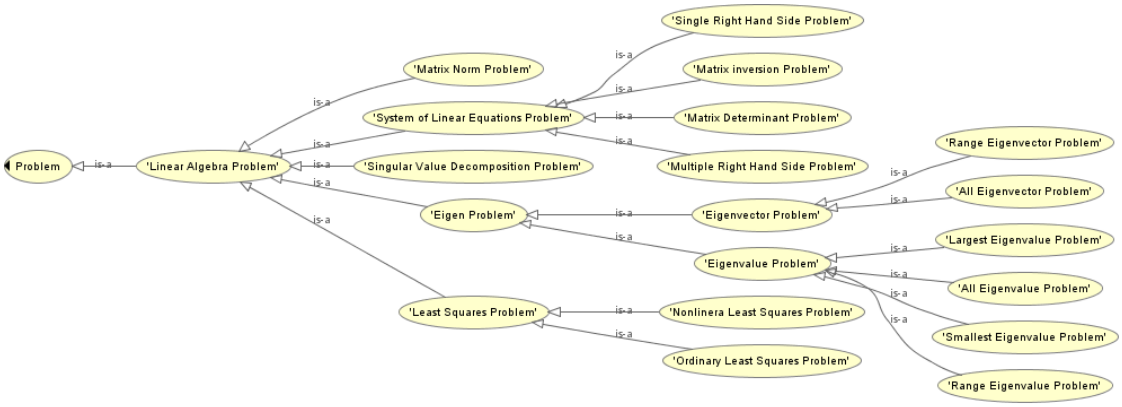

FIGURE 3. Hierarchy of problems in *AiG Expert Ontology*

Hierarchy of algorithms

Figure 4 shows the first two levels of the hierarchy of classes representing algorithms / methods that can be used to solve problems from Figure 3 hierarchy, for a given input data. This part of the ontology, along with instances of the *Job Profile* and the *Expert Opinion* classes, is most crucial and is currently being developed on the basis of the domain expert knowledge. In other words, here we depict only a preliminary representation, which is fairly complete, but is expected to be further refined.

FIGURE 4. Part of the hierarchy of algorithms in *AiG Expert Ontology*

The algorithms are divided into five main classes that can be again decomposed, depending on how specific the expert, or the Grid user, wants to be:

- *Solution Accuracy Improvement* – classes representing methods to improve accuracy of the numerical solutions of the systems of linear equations.
- *Transformation* – classes representing commonly used transformations that are composed of more granular operations and are used within certain algorithms.
- *Eigen Solver* – classes representing methods for solving eigen problems.
- *Fundamental Operation* – classes representing hierarchy of core operations in the area of computational linear algebra. These classes are based on the classical BLAS and LAPACK libraries.
- *Linear Equation Solver* – classes representing hierarchy of algorithms for solving systems of linear equations, further decomposed into direct (for more details see Figure 5) and iterative solvers (depicted in Figure 6).

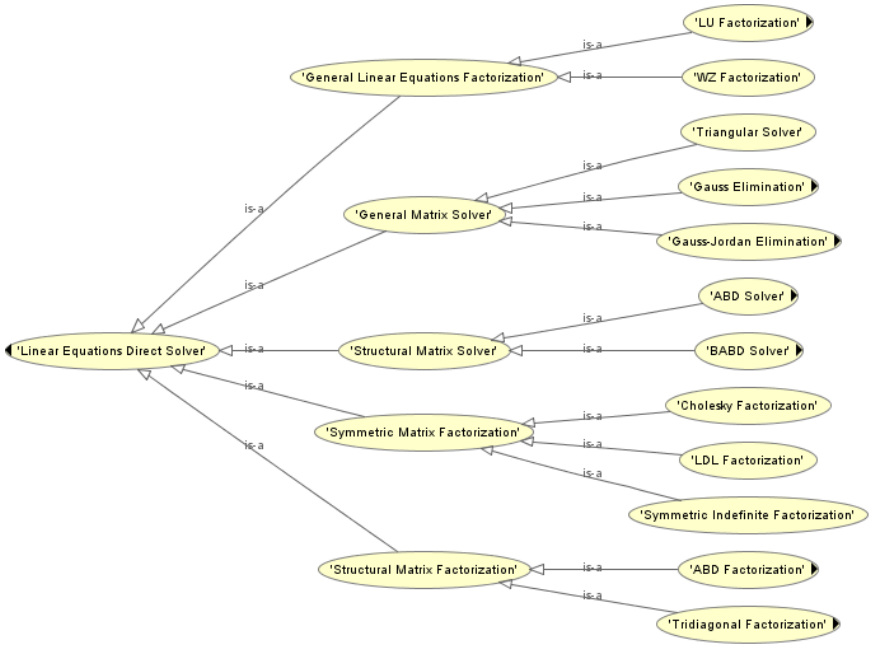

FIGURE 5. Part of the hierarchy of linear equations direct solvers in *AiG Expert Ontology*

FIGURE 6. Part of the hierarchy of linear equations iterative solvers in *AiG Expert Ontology*

Hierarchy of data properties

To describe characteristics of input data, we utilize *Data Element* and *Data Property* classes (see Figure 7) and the property *hasProperty* that defines their relationship. For the computational linear algebra we focus on the *Matrix* subclass of the *Data Element* and the *Matrix Property* subclass of the *Data Property* that is also a superclass for a

hierarchy of properties that describe the matrix (e.g. symmetricity, density, structure, etc.).

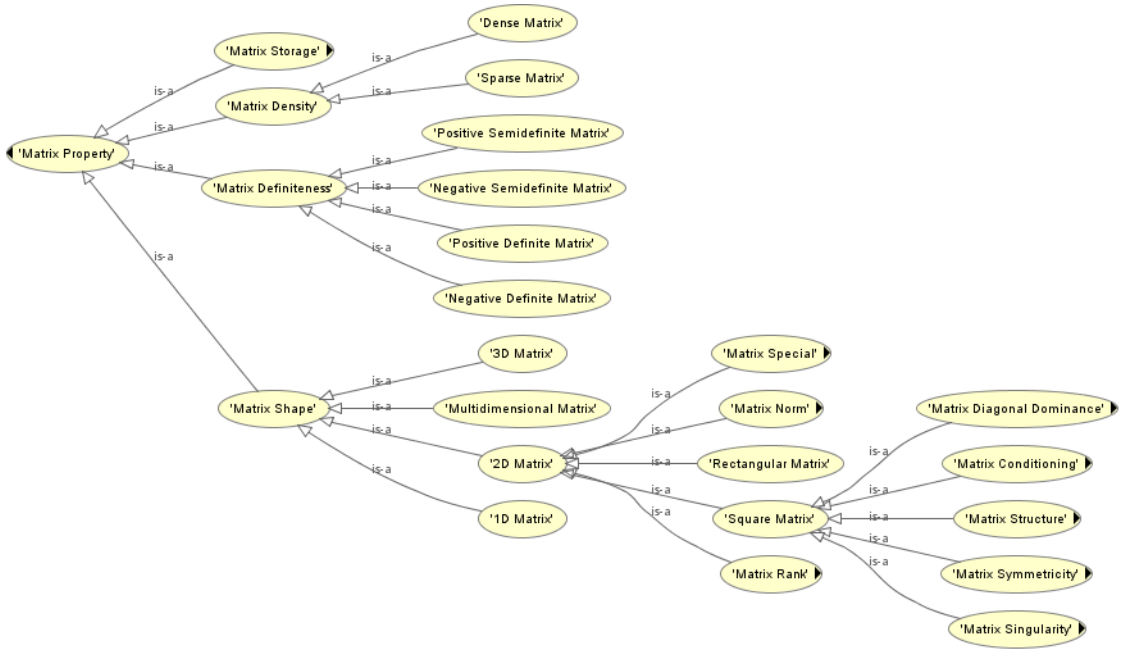

FIGURE 7. Part of the hierarchy of data (matrix) properties in *AiG Expert Ontology*

Note that properties concerning matrix structure can be further decomposed, as in Figure 8. Obviously, Figures 7 and 8 do not show the full hierarchy of already available properties.

EXAMPLE

Let us now follow with an example of how the expert knowledge can be stored and extracted from the ontology using the conceptual model presented in the previous section.

From user's perspective, the crucial part is to specify the job profile and the resource requirements for the job to be executed. In the *AiG* project all user interaction with the system are completed using a dynamic ontology-based web interface (for more details, see [13]). Here, the user can specify only such requirements for resources needed to execute the job that are available within the system – instantiated in the ontology (assuming that she is certain about her needs and does not require expert system support). However, she can also "underdetermine" the request, by selecting an instance of a subclass of the *Problem* class, e.g. the *System Of Linear Equations Problem*. The job profile instance may optionally have properties: (i) *forData* that specify the input, e.g. an instance of the class *Matrix*, with values of *hasProperty* being instances of the *Positive Definite Matrix* and the *Symmetric Matrix* classes, and (ii) *hasAlgorithm* that specifies an iterative solver, e.g. an instance of the class *Linear Equation Iterative Solver* to be used to solve the problem (here, it is assumed that the user does not have knowledge, which iterative method to use) see Figure 6 and the snippet, below.

```
<equivalentClass><Class>
    <intersectionOf rdf:parseType="Collection">
        <rdf:Description rdf:about="http://gridagents.sourceforge.net/AiGExpertOntology#
            JobProfile"/>
        <Restriction>
            <onProperty rdf:resource="http://gridagents.sourceforge.net/AiGExpertOntology#forData
                "/>
            <someValuesFrom><Class><intersectionOf rdf:parseType="Collection">
            <rdf:Description rdf:about="http://gridagents.sourceforge.net/AiGExpertOntology#
                Matrix"/>
            <Restriction>
                <onProperty rdf:resource="http://gridagents.sourceforge.net/AiGExpertOntology#
                    hasProperty"/>
```

FIGURE 8. Part of the hierarchy of matrix structure in *AiG Expert Ontology*

```
            <someValuesFrom><Class><intersectionOf rdf:parseType="Collection">
                <rdf:Description rdf:about="http://gridagents.sourceforge.net/
                    AiGExpertOntology#PositiveDefiniteMatrix"/>
            </intersectionOf></Class></someValuesFrom>
        </Restriction>
        <Restriction>
            <onProperty rdf:resource="http://gridagents.sourceforge.net/AiGExpertOntology#
                hasProperty"/>
            <someValuesFrom><Class><intersectionOf rdf:parseType="Collection">
                <rdf:Description rdf:about="http://gridagents.sourceforge.net/
                    AiGExpertOntology#SymmetricMatrix"/>
            </intersectionOf></Class></someValuesFrom>
        </Restriction>
        </intersectionOf></Class></someValuesFrom>
    </Restriction>
    <Restriction>
        <onProperty rdf:resource="http://gridagents.sourceforge.net/AiGExpertOntology#
            forProblem"/>
        <someValuesFrom><Class><intersectionOf rdf:parseType="Collection">
            <rdf:Description rdf:about="http://gridagents.sourceforge.net/AiGExpertOntology#
                SystemOfLinearEquationsProblem"/>
        </intersectionOf></Class></someValuesFrom>
    </Restriction>
    <Restriction>
```

```
            <onProperty rdf:resource="http://gridagents.sourceforge.net/AiGExpertOntology#
                hasAlgorithm"/>
            <someValuesFrom><Class><intersectionOf rdf:parseType="Collection">
            <rdf:Description rdf:about="http://gridagents.sourceforge.net/AiGExpertOntology#
                LinearEquationIterativeSolver"/>
            </intersectionOf></Class></someValuesFrom>
        </Restriction>
    </intersectionOf>
</Class></equivalentClass>
```

On the other hand, in the *AiG Expert ontology* we store respective instances to be matched with what the user specifies. Note, that (due to the space limitations) this is a very simplified example, in which not all available properties are specified to conclusively adjust the algorithm and the resource proposed by expert. Below we include the ontology snippet representing job profile with the related expert recommendation.

```
<owl:NamedIndividual rdf:about="&AiGExpertInstances;jobProfile1">
    <rdf:type rdf:resource="&AiGExpertOntology;JobProfile"/>
    <AiGExpertOntology:hasExpertOpinion rdf:resource="&AiGExpertInstances;expertOpinionIndiv1"/>
    <AiGExpertOntology:forProblem rdf:resource="&AiGExpertInstances;lseProblem1Indiv"/>
    <AiGExpertOntology:hasAlgorithm rdf:resource="&AiGExpertInstances;lseSolverIndiv1"/>
    <AiGExpertOntology:forData rdf:resource="&AiGExpertInstances;matrix1Indiv"/>
</owl:NamedIndividual>
<owl:NamedIndividual rdf:about="&AiGExpertInstances;expertOpinionIndiv1">
    <rdf:type rdf:resource="&AiGExpertOntology;ExpertOpinion"/>
    <AiGExpertOntology:proposedByExpert rdf:resource="&AiGExpertInstances;HPCExpert1"/>
    <AiGExpertOntology:hasRecommendedResource rdf:resource="&AiGExpertInstances;
        requiredResourceIndiv1"/>
</owl:NamedIndividual>
```

Respective requirements regarding e.g. matrix properties and the algorithm that the job profile in the *AiG Expert ontology* is referring to, are represented with the class expressions and instances in the following code snippet.

```
<owl:Class rdf:about="&AiGExpertInstances;Matrix1">
    <owl:equivalentClass><owl:Class>
        <owl:intersectionOf rdf:parseType="Collection">
            <rdf:Description rdf:about="&AiGExpertOntology;Matrix"/>
            <owl:Restriction>
                <owl:onProperty rdf:resource="&AiGExpertOntology;hasProperty"/>
                    <owl:someValuesFrom><owl:Class><owl:intersectionOf rdf:parseType="Collection"
                        >
                        <rdf:Description rdf:about="&AiGExpertOntology;PositiveDefiniteMatrix"/>
                    </owl:intersectionOf></owl:Class></owl:someValuesFrom>
             </owl:Restriction>
            <owl:Restriction>
                <owl:onProperty rdf:resource="&AiGExpertOntology;hasProperty"/>
                <owl:someValuesFrom><owl:Class><owl:intersectionOf rdf:parseType="Collection">
                    <rdf:Description rdf:about="&AiGExpertOntology;SymmetricMatrix"/>
                </owl:intersectionOf></owl:Class></owl:someValuesFrom>
             </owl:Restriction>
        </owl:intersectionOf>
    </owl:Class></owl:equivalentClass>
</owl:Class>
<owl:NamedIndividual rdf:about="&AiGExpertInstances;matrix1Indiv">
    <rdf:type rdf:resource="&AiGExpertInstances;Matrix1"/>
</owl:NamedIndividual>
<owl:Class rdf:about="&AiGExpertInstances;LSEIterStatSolver1">
    <owl:equivalentClass><owl:Class>
        <owl:unionOf rdf:parseType="Collection">
            <rdf:Description rdf:about="&AiGExpertOntology;GaussSeidelMethod"/>
            <rdf:Description rdf:about="&AiGExpertOntology;JacobiMethod"/>
        </owl:unionOf>
    </owl:Class></owl:equivalentClass>
</owl:Class>
<owl:NamedIndividual rdf:about="&AiGExpertInstances;lseSolverIndiv1">
    <rdf:type rdf:resource="&AiGExpertInstances;LSEIterStatSolver1"/>
</owl:NamedIndividual>
```

The system analyzes the ontologies and uses the expert knowledge from the indicated problem domain to suggest: (i) what is the optimal algorithm / method to solve the problem, (ii) what resource configuration is recommended by the experts. Let us assume, that for a given input data and problem, there exists one expert recommendation to use resource with the configuration specified as the *requiredResourceIndiv1*, and the user would be informed of two algorithm alternatives (Jacobi method, and Gauss-Seidel method). Obviously, there can be a situation were more than one expert opinion can be associated with a given job profile. In such a case, in the *AiG* system, the Analytical Hierarchy Process (*AHP*; [14]) method for multicriterial decision making is applied to select the optimal recommendation and / or prepare additional suggestions how to extend / modify the resource requirement(s). The way to combine ontologies and the *AHP* method was introduced in [15, 16].

CONCLUDING REMARKS

The aim of this paper is to introduce an approach selected to apply domain knowledge to support users in the *AiG* Grid management system. Moreover, an ontology used to store expert knowledge in the selected initial area of application (computational linear algebra domain) was described in terms of its structure and possible semantic processing. Our current goal is to continue the development of the ontology of computational linear algebra, and test its application in a prototype of the user decision support subsystem. The key issue that has to be addressed, to efficiently help the user, is to represent (in the domain ontology) as much of expert knowledge as possible, and this is an ongoing process of ontology extension. Note that we do not have to deal with conflicting expert knowledge, as we can represent knowledge of each individual expert as an ontology instance. Then, as mentioned above, we will apply the AHP method to combine their recommendations to create the "ultimate one" to be presented to the user. Our future work should also concentrate on assisting users with specifying properties of input data, since, for instance, some of the properties of the matrix can be automatically recognized (with a set of rules) on the basis of such parameters. Finally, the outstanding issue of submitting a job to "any" middleware has to be approached.

ACKNOWLEDGMENTS

Work presented here is a part of the Poland-Bulgaria collaborative grant "Parallel and distributed computing practices."

REFERENCES

1. M.Dominiak, W.Kuranowski, M.Gawinecki, M.Ganzha, and M.Paprzycki, "Utilizing agent teams in grid resource management—preliminary considerations," in *Proc. of the IEEE J. V. Atanasoff Conference*, IEEE CS Press, Los Alamitos, CA, 2006, pp. 46–51.
2. W.Kuranowski, M.Ganzha, M.Gawinecki, M.Paprzycki, I.Lirkov, and S.Margenov, *International Journal of Computational Intelligence Research* **4**, 9–16 (2008).
3. K.Wasielewska, M.Drozdowicz, M.Ganzha, M.Paprzycki, N.Attaui, D.Petcu, C.Badica, R.Olejnik, and I.Lirkov, "Negotiations in an Agent-based Grid Resource Brokering Systems," in *Trends in Parallel, Distributed, Grid and Cloud Computing for Engineering*, edited by P. Ivanyi, and B. Topping, Saxe-Coburg Publications, Stirlingshire, UK, 2011.
4. K. Łysik, M. Ganzha, K. Wasielewska, M. Paprzycki, J. Brennan, V. Holmes, and I. Kureshi, "Combining AiG Agents with Unicore grid for improvement of user support," in *Proceedings of the First International Symposium on Computing and Networking – Across Practical Development and Theoretical Research*, 2013 submitted for publication.
5. Ian Foster, Carl Kesselman, Steven Tuecke, The anatomy of the grid – enabling scalable virtual organizations (2001), URL `http://arxiv.org/abs/cs/0103025`.
6. EMI, MNA3.2 Open Source Software Initiative (2013), URL `http://cds.cern.ch/record/1450878/files/EMI-MNA3.2-1450878-OS_Initiative_Charter_v1.0.pdf?version=1`, EMI Collaboration.
7. M.Lucks, *A Knowledge-Based Framework for the Selection of Mathematical Software*, Ph.D. thesis, Southern Methodist University (1990).
8. D.Petcu, and V.Negru, "Interactive system for stiff computations and distributed computing," in *Proceedings of IMACS'98: International Conference on Scientific Computing and Mathematical Modelling*, IMACS, 1998, pp. 126–129.
9. P. Bunus, "A Simulation and Decision Framework for Selection of Numerical Solvers in Scientific Computing," in *Proceedings of the 39th annual Symposium on Simulation*, ANSS '06, IEEE Computer Society, Washington, DC, USA, 2006, pp. 178–187.
10. S. Weerawarana, E. N. Houstis, J. R. Rice, A. Joshi, and C. E. Houstis, *ACM Trans. Math. Softw.* **22**, 447–468 (1996).
11. M. Kamel, W. Enright, and K. Ma, *ACM Trans. Math. Softw.* **19**, 44–62 (1993).

12. M.Drozdowicz, K.Wasielewska, M.Ganzha, M.Paprzycki, N.Attaui, I.Lirkov, R.Olejnik, D.Petcu, and C.Badica, "Ontology for Contract Negotiations in Agent-based Grid Resource Management System," in *Trends in Parallel, Distributed, Grid and Cloud Computing for Engineering*, edited by P.Ivanyi, and B.H.V.Topping, Saxe-Coburg Publications, Stirlingshire, UK, 2011.
13. M. Drozdowicz, M. Ganzha, M. Paprzycki, P. Szmeja, and K. Wasielewska, "OntoPlay - A Flexible User-Interface for Ontology-based Systems," in *AT*, 2012, pp. 86–100.
14. T.L.Saaty, *European Journal of Operational Research* **48**, 9–26 (1990).
15. K.Wasielewska, and M.Ganzha, "Using analytic hierarchy process approach in ontological multicriterial decision making - Preliminary considerations," in *AIP Conference Proceedings*, American Institute of Physics, 2012, vol. 1487/1, pp. 95–103.
16. K. Wasielewska, M. Ganzha, M. Paprzycki, P. Szmeja, M. Drozdowicz, I. Lirkov, and C. Badica, *Information Technology And Control* (2014 submitted for publication).