Semantic technologies in a decision support system

K. Wasielewska, M. Ganzha, M. Paprzycki, C. Bădică, M. Ivanovic, and I. Lirkov

Citation: AIP Conference Proceedings **1684**, 060002 (2015); doi: 10.1063/1.4934301
View online: http://dx.doi.org/10.1063/1.4934301
View Table of Contents: http://scitation.aip.org/content/aip/proceeding/aipcp/1684?ver=pdfcov
Published by the AIP Publishing

Semantic Technologies in a Decision Support System

K. Wasielewska[1,a)], M. Ganzha[1,2], M. Paprzycki[1,3], C. Bădică[4], M. Ivanovic[5] and I. Lirkov[6]

[1]*Systems Research Institute Polish Academy of Sciences, Warsaw, Poland*
[2]*Warsaw University of Technology, Warsaw, Poland*
[3]*Warsaw Management Academy, Warsaw, Poland*
[4]*Department of Computer Science, University of Craiova, Craiova, Romania*
[5]*Department of Mathematics and Informatics, Faculty of Sciences, University of Novi Sad, Novi Sad, Serbia*
[6]*Institute of Information and Communication Technologies, Bulgarian Academy of Sciences, Sofia, Bulgaria*

[a)]Corresponding author: katarzyna.wasielewska@gmail.com

Abstract. The aim of our work is to design a decision support system based on ontological representation of domain(s) and semantic technologies. Specifically, we consider the case when Grid / Cloud user describes his/her requirements regarding a "resource" as a class expression from an ontology, while the instances of (the same) ontology represent available resources. The goal is to help the user to find the best option with respect to his/her requirements, while remembering that user's knowledge may be "limited." In this context, we discuss multiple approaches based on semantic data processing, which involve different "forms" of user interaction with the system. Specifically, we consider: (a) ontological matchmaking based on SPARQL queries and class expression, (b) graph-based semantic closeness of instances representing user requirements (constructed from the class expression) and available resources, and (c) multicriterial analysis based on the AHP method, which utilizes expert domain knowledge (also ontologically represented).

INTRODUCTION

The context of our work is provided by the Agents in Grid (AiG; [1, 2, 3]) project, which proposed an agent-semantic infrastructure for efficient resource management in the Grid. The system is based on the following assumptions (practical application of this approach has been described in [4, 5]):

- Grid is treated as an open environment that can be easily joined and used
- among Grid "users" we distinguish:
  - resource providers – that offer resources and are "paid" for their use;
  - resource consumers – commission job execution and "pay" for resource utilization;
  - here, "payment" may mean "abstract payment," as well as actual payment involving monetary units.
- system utilizes software agents and semantic data processing; all users and resources are represented by software agents, while all "knowledge" is captured and represented in the form of ontologies;
- agents representing resources work in teams (user negotiates and commissions job execution to a team of agents; note that an individual resource can be treated as an one-member team);
- a CIC component stores (and delivers) information about the Grid structure and available teams/resources;
- Grid users may have different level of expertise, *i.e.*, they may know the nature of the job (problem to be solved) and input data but do not know what is the optimal resource configuration and / or tool(s) (application/library) that is available and should be used.

The need to develop a decision support subsystem is based on the fact that Grid users can be roughly divided into: standard (majority) and active (few) (see, [4]). Active users explore new possibilities and methods (in terms of hardware/tools/libraries/applications) available to solve their problems. On the other hand, standard users are interested only in solving specific problems and do not devote time to learn about the computing infrastructure (they know

Application of Mathematics in Technical and Natural Sciences
AIP Conf. Proc. 1684, 060002-1–060002-12; doi: 10.1063/1.4934301

only one way to solve a given problem). Standard users, typically, have minimal knowledge concerning computers, *e.g.*, artists who use 3D rendering programs. This results in: (i) communication gap between creators of new methods and users, (ii) continuous use of one approach to solve a given problem even though (a) better alternative(s) exists.

Decision support subsystem should help users (with different level of expertise) to choose optimal resources to solve their problems, and to select the best contract defining terms of use of these resources. Note that what follows naturally generalizes to multiple scenarios (where decisions have to be made in a situation when the "domain" is ontologically represented). For instance, thus stated problem naturally arises in Cloud Computing (*i.e.*, the selection of appropriate resources). Here, the main difference would be use of Cloud-ontologies, instead of Grid ontologies.

User support scenario

Let us now briefly outline the scenario in which the user wants to utilize the Grid resource (in the AiG system):

1. User specifies resource requirements (possibly job description) with ontology-driven web interface (OntoPlay; [6]);
2. *User_agent* (agent representing the user), queries the CIC for the list of teams with resources matching these criteria (decision support step 1);
3. *User_agent* eliminates from the list teams that are not trustworthy;
4. User specifies requirements regarding the contract (describes terms of collaboration with resource provider; via the OntoPlay interface);
5. *User_agent* communicates with "resource teams" and sends contract proposals;
6. *User_agent* evaluates contract offers and selects one if the terms are acceptable (decision support step 2).

When considering the support steps 1 and 2, we have identified the following issues:

- in both steps selection is based on multiple criteria; furthermore, decision support module should take advantage of the fact that criteria can have assigned priority weights expressing user preferences;
- set of criteria used to describe resources/contact offer can vary; however, the ontology-driven interface assures that the description is always consistent with the ontology that defines all possible criteria (corresponding to properties of appropriate concepts);
- ontology organizes the criteria structure into a hierarchy (concepts/classes map to nodes and properties map to edges; the roots in classes *ComputingElement* or *JobExecutionConditions*, corresponding to the resource and the contract in steps 1 and 2 respectively);
- the system includes an expert ontology; here, opinions of multiple experts from a given domain can be thus used to assess user requirements regarding resource/contract; note that experts can have different weights, *e.g.*, opinions of a student and a professor are not equally important; moreover, experts can have different level of knowledge regarding different problems from the domain;
- in step 1, *user_agent* should obtain list of resources matching user's requirements; this step can be extended so that the support module confronts user requirements with one selected (most representative) expert opinion ("representing" all expert opinions for a given problem) in order to improve user's requirements specification;
- in step 2, support module should help selecting the best contract, from all offers received from "resource teams".

After analysis of the design of the system, the use case scenario and the identified issues concerning needed support, we have selected methods originating from semantic technologies that can be applied to provide the needed functionality. Before describing them, let us briefly consider ontologies designed for the AiG system (and their use).

Ontologies in the AiG system – outline

Ontology is a formal, explicit description of concepts in the domain, and relationships between them that provide common understanding of the structure of information and its usage rules and allows to analyze and reuse domain knowledge [7]. Main elements of an ontology are classes (concepts), properties and restrictions. Ontology can be designed and analyzed within: (i) T-Box – the conceptual model, *i.e.*, set of concepts and their properties; (ii) A-Box – assertions describing facts within vocabulary defined in T-Box. The key ontologies in the AiG system are (see, [6, 8]):

- AiG Grid Ontology – concepts used to describe Grid structure and resource configuration;
- AiG Conditions Ontology – concepts regarding terms of collaboration between user and resource provider;
- AiG Expert Ontology – representing experts' knowledge about a given domain.

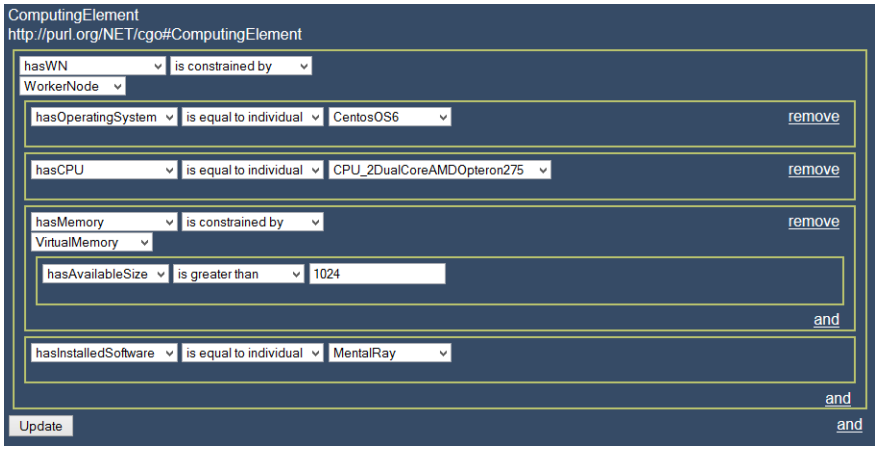

FIGURE 1: Ontology-driven web interface

The key module in the system is the ontology-driven web interface (OntoPlay; [9]). It allows the user without knowledge about the ontological constructs to specify requirements regarding resources and contract. These requirements are then automatically transformed into class expressions, used in the system. Figure 1 shows the web interface that is dynamically generated from the AiG ontologies. The *ComputingElement* is the class (concept) that is being described. Each combo box contains properties defined for the domain class. Depending on the selection from the combo box, user is presented with operators and values that can be used as ranges. Requirements can be nested, *e.g.*, in case of object property nested combo boxes for a range class are generated. Figure shows an example of the required resource that should have operating system $Centos6$, CPU being an individual $CPU_2DualCoreAMDOpteron275$ (defined in the ontology), virtual memory with available size greater than $1024MB$ and installed the *MentalRay* software. Analogically, user specifies requirements for the contract and the description of the job to be executed.

ANALYSIS OF SELECTED APPROACHES TO USER SUPPORT

When designing the user support module, taking into account issues identified in Section a) we have selected several approaches with different level of complexity and matching different situations. Let us stress, once again, that the methods used in the AiG system *must* be compliant with the assumption that *all* data in the system is ontologically represented. Henceforth, any MCA method that, for instance, forces one to "flatten" the hierarchical structure of the ontology (assumes that all criteria are on the same level), to be applied, is not considered (see, also [10]).

SPARQL queries

The first (simplest) approach, can be used only in step 1, *i.e.*, selecting resources that exactly match user requirements (contract selection typically involves a ranked list of offers). It is based on use of the SPARQL ([11]) – semantic SQL-like query language used for the RDF demarcated data ([12]). Note that the AiG ontologies are specified using the OWL language ([13]) and using RDF does not allow to express all constructs. SPARQL considers only the RDF, while querying the ontology, *e.g.*, the *subClassOf* transitivity is not understood.

Each SPARQL request starts with the PREFIXes section (specifying the namespaces). Identifiers beginning with question mark "?" denote variables. The following code snippet is an example of a SPARQL query that returns individuals being instances of a *ComputingElement* class that have resources with configuration corresponding to requirements specified in Figure 1.

```
PREFIX aig: <http://gridagents.sourceforge.net/AiGGridOntology#>
PREFIX aiginst: <http://gridagents.sourceforge.net/AiGGridInstances#>
PREFIX cgo:  <http://purl.org/NET/cgo#>
select ?res
where {
?res a aig:ComputingElement .
        cgo:hasWorkerNode ?wn .
        ?wn cgo:hasOperatingSystem aiginst:CentosOS6 .
        ?wn cgo:hasCPU aiginst:CPU_2DualCoreAMDOpteron275 .
        ?wn cgo:hasMemory ?memory .
```

```
        ?memory cgo:hasAvailableSize ?size .
filter ( ?size > 1024 )
}
```

Here, the decision support capabilities of the system are very limited. With the ontologically-driven web interface user enters restrictions regarding the resource (see, Figure 1). Resource is represented by the instance of the *ComputingElement*, with data properties having values specified with *is less than*, *is greater than*, *is equal to* relations, and object properties with values specified using *is equal to individual* or *is constraint by* relations (which may lead to nested restrictions). Resource requirements are transformed (by the OntoPlay) into a class expression that is passed to the *user_agent* and later to the CIC component. The CIC transforms class expression into a SPARQL query and executes it. The transformation generates the *where* condition for each restriction in the class expression. When the restrictions are nested, the "?" variable names are generated to include these restrictions into the *where* phrase. When the comparison operators are used the "q" *filter* section is added. The resulting list of resources is *not* ranked, since *all* criteria are treated as equally important, and the SPARQL query filters resources that exactly match the criteria.

FIGURE 2: User requirements for resource

Figure 2 shows example of requirements for the resource specified by the user via the OntoPlay interface. Here, user wants to execute a job on a resource that has Debian operating system (*debian_5.0* individual), virtual memory with available size greater than 1024 MB, available storage space greater than 2014 MB and EXT3 (*ext3* individual) file system. The following code snippet shows class expression representing the above user requirements (generated by the web interface and passed to *user_agent*; note that namespaces were shortened to save space):

```
<owl:equivalentClass>
  <owl:Class>
    <owl:intersectionOf rdf:parseType="Collection">
      <rdf:Description rdf:about="cgo#ComputingComponent"/>
      <owl:Restriction>
        <owl:onProperty rdf:resource="aig#hasMemory"/>
        <owl:someValuesFrom>
          <owl:Class>
            <owl:intersectionOf rdf:parseType="Collection">
            <rdf:Description rdf:about="aig#VirtualMemory"/>
            <owl:Restriction>
            <owl:onProperty rdf:resource="aig#hasAvailableSize"/>
            <owl:someValuesFrom>
            <rdfs:Datatype>
            <owl:onDatatype rdf:resource="XMLSchema#integer"/>
            <owl:withRestrictions rdf:parseType="Collection">
            <rdf:Description>
            <xsd:minExclusive rdf:datatype="XMLSchema#integer">1024<xsd:minExclusive>
            </rdf:Description>
                        </owl:withRestrictions>
                        </rdfs:Datatype>
                        </owl:someValuesFrom>
              </owl:Restriction>
            </owl:intersectionOf>
          </owl:Class>
        </owl:someValuesFrom>
      </owl:Restriction>
      <owl:Restriction>
        <owl:onProperty rdf:resource="aig#hasStorageSpace"/>
        <owl:someValuesFrom>
          <owl:Class>
            <owl:intersectionOf rdf:parseType="Collection">
            <rdf:Description rdf:about="cgo#StorageSpace"/>
            <owl:Restriction>
            <owl:onProperty rdf:resource="aig#hasFileSystem"/>
                <owl:hasValue rdf:resource="aiginst#ext3"/>
            </owl:Restriction>
```

```
            <owl:Restriction>
            <owl:onProperty rdf:resource="aig#hasAvailableSize"/>
            <owl:someValuesFrom>
            <rdfs:Datatype>
            <owl:onDatatype rdf:resource="XMLSchema#integer"/>
            <owl:withRestrictions rdf:parseType="Collection">
            <rdf:Description>
            <xsd:minExclusive rdf:datatype="XMLSchema#integer">2014<xsd:minExclusive>
                    </rdf:Description>
            </owl:withRestrictions>
            </rdfs:Datatype>
            </owl:someValuesFrom>
              </owl:Restriction>
            </owl:intersectionOf>
          </owl:Class>
        </owl:someValuesFrom>
      </owl:Restriction>
      <owl:Restriction>
        <owl:onProperty rdf:resource="aig#isRunningOS"/>
        <owl:hasValue rdf:resource="aiginst#debian_5.0"/>
      </owl:Restriction>
    </owl:intersectionOf>
  </owl:Class>
</owl:equivalentClass>
</owl:Class>
```

Note that the *Restriction* nodes in the above XML snippet correspond to the restrictions from the OntoPlay interface. The *onProperty* node specifies what attribute is restricted, and the next node defines the restriction. The following code snipped shows the resultant SPARQL query after the transformation done by the system:

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs:<http://www.w3.org/2000/01/rdf-schema#>
PREFIX cgo:<http://purl.org/NET/cgo#>
PREFIX aig:<http://gridagents.sourceforge.net/AiGGridOntology#>
PREFIX cgoinst:<http://purl.org/NET/cgoInstances#>
PREFIX aiginst:<http://gridagents.sourceforge.net/AiGGridInstances#>
SELECT ?x
WHERE {
    ?x rdf:type cgo:ComputingElement .
    ?x aig:hasStorageSpace ?x1 .
    ?x aig:isRunningOS aiginst:debian_5.0 .
    ?x aig:hasMemory ?x2 .
    ?x1 aig:hasFileSystem aiginst:ext3 .
    ?x1 aig:hasAvailableSize ?x1f .
    ?x2 aig:hasAvailableSize ?x2f .
FILTER ( ?x1f > 2014 &&
     ?x2f > 1024 )
}
```

As a result, the *user_agent* receives a list of resources that match the requirements – resources that have Debian operating system (*debian_5.0* individual), virtual memmory with available size greater than 1024 MB, available storage space greater than 2014 MB and EXT3 (*ext3* individual) file system. Note that all requirements must be met (*exactly*) and the result presented to the user is not ordered. Hence, the user does not receive any indication, which resource is "the best" / "better than others" (all resources are equally good). In case when no resource matches the requirements, user is presented with appropriate information and he should modify the input requirements, *e.g.*, relax the restrictions.

Class expression matchmaking

Another approach is to utilize class expression matching mechanism for the resource selection (step 1). This approach is similar to the SPARQL query-based approach. It also does not allow one to take advantage of "weighting of properties" and returns an unordered list of resources exactly matching the request. The overall user support scenario does not change but the matching process is different. User specifies requirements via web interface as in Figures 1 and 2. These are transformed into class expression (see code snippet in the previous section). Next, the reasoner (embedded in CIC component) analyzes the ontology and matches individuals representing resources to the class expression specifying restrictions on a given class instances.

Reasoners are a crucial component for working with OWL ontologies. They allow to query an OWL ontology and deduce knowledge that is not explicitly expressed (has more expressiveness than the RDF data). Most popular reasoners include: HermiT, Pellet, FACT++, JFact, RacerPro.

Lets consider the following example – in the SPARQL-based approach found resources were instances of the *ComputingElement* class, and only instances of this class could match the query. In the class expression matchmaking, also resources that are instances of the *ComputingElement* subclasses, *e.g.*, the *StorageElement* or classes declared as equivalent to the *ComputingElement*, *e.g.*, the *HPComputingElement* can be returned. This makes the query structure more flexible.

Let us assume that the SPARQL query returns two resources (*resource1* and *resource2*) that are both instances of the *ComputingElement* class and match the requirements. In the system ontology there may exist *resource3* that

also matches the requirements but is an instance of the *HPCComputingElement* (subclass of *ComputingElement*). Class expression matchmaking will include also *resource3* in the result set – the reasoner will deduce that instances of classes of the searched class should also be considered. The *resource3* cannot be captured by the SPARQL query.

As above, the results are not ranked and all criteria are treated as equally important. In both cases, class expression matching and the SPARQL approach, if no resource matches the requirements, the user is presented with appropriate information and (s)he should modify the serach criteria, *e.g.*, relax the restrictions.

Graph-based ontological matchmaking

This approach uses graph-based ontological matchmaking algorithm described in [14, 15]. It can be applied in both step 1 and step 2 because the resultant lists of resources or contract offers is ranked. Furthermore, it allows one to express importance of individual aspects of the proposal.

The approach is based on the fact that an ontology can be represented as a directed graph where concepts/individuals are nodes and edges represent properties (node that properties map to decision criteria). Graph can be generated on both conceptual (T-Box) and individuals (A-Box) levels. On the conceptual level, nodes are classes (concepts) and edges correspond to properties defined in the ontology. On the individuals level nodes correspond to individuals and edges to properties asserted for considered individuals. Matchmaking becomes measuring "distance" between instances of concepts (classes) in the ontology. Assumptions for the matchmaking algorithm are:

- having more relations from one object to another means that they are closer (more relevant to each other);
- each relation has different importance depending on the type of relation (relations on conceptual level can be weighted according to their semantic importance; while the default weight is 1);
- if relations are of the same type, the "weight" of the connection can vary between individual objects (instances).

This approach allows one to capture: user preferences and expert knowledge. Expert knowledge is considered on the conceptual level, where each edge can be assigned a "distance value" that is inversely proportional to relevance between the two adjacent nodes. Here, Experts can decide, for instance, that for computational jobs the *hasInstalled-Software* property is more important than the *hasCPU* property. User preferences (based on user's expertise) are represented by assigning weights to the edges (scaling edges) on the level of individuals. An example of user support in resource selection is as follows:

- user specifies (via the OntoPlay interface) requirements for resource needed to execute a given job (the more complete the specification the "better"); *e.g.*, at least 512 MB memory (1024 preferred), Debian Linux in version 5.0 preferred;
- *user_agent* generates the *representative instance* of requirements (changing preferred into exact values);
- the CIC component uses a graph-based matchmaking algorithm to calculate distance between the representative instance and instances representing available resources;
- result is a list of resources ranked with respect to the degree of relatedness.

Figure 3 shows part of a conceptual level model generated from the AiG Grid Ontology. Concepts (classes) are nodes and properties are directed associations. Each edge has assigned distance value (the greater the value the less important is relation between objects; value that should be assigned by experts; applied across the algorithm). Note that, here, the software available on the computing node is the most important. Distance values equal to 1 are default.

Figure 4 shows individuals corresponding to classes from the conceptual level. The *reqComputingElement* is an example individual generated from user requirements. Individuals *resource1* and *resource2* represent resources available in the system. The *resource1* has the required *blas3.5* installed and Debian operating system but in version 4.0 instead of required 5.0. The *resource2* has the required operating system *debian5.0* and the required CPU *cpu1*.

The CIC component uses the algorithm from [14] to calculate the distance between *reqComputingElement* and *resource1* and *resource2*, respectively. Note that user has possibility to assign weights to properties in the individual level, *e.g.*, assign weight 1.5 to *hasCPU* property. In this example, *resource1* is "closer" than *resource2* to the individual representing requirements.

The algorithm returns resources for which exists at least one path from individual representing requirements. if there are no such resources then user is informed and needs to modify requirements.

The graph-based approach is dedicated to user's with in-depth knowledge about methods/libraries/applications that they want to use to solve a given problem. To fully benefit from it, they should assign priorities to properties (mapped onto criteria). Furthermore, expert knowledge (if available) can be used to improve the matching algorithm.

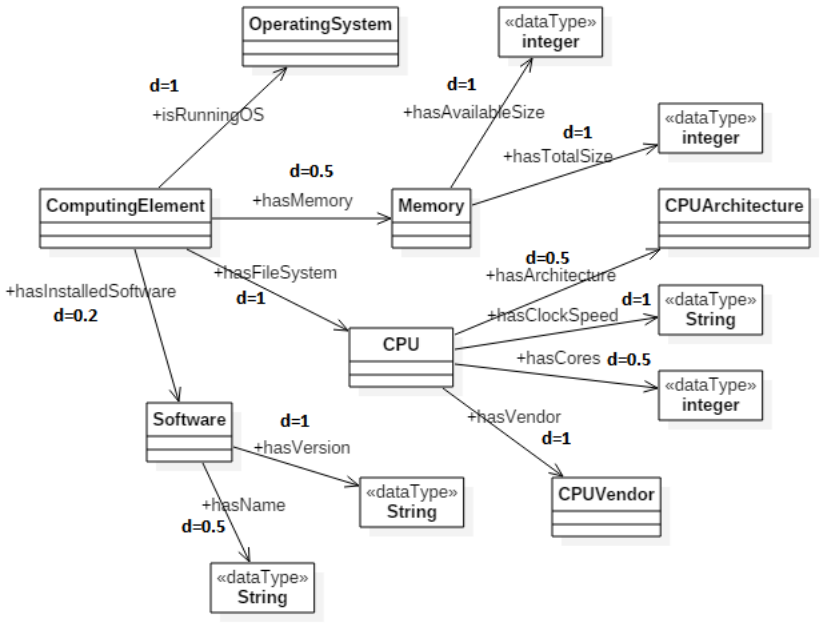

FIGURE 3: Fragment of conceptual model

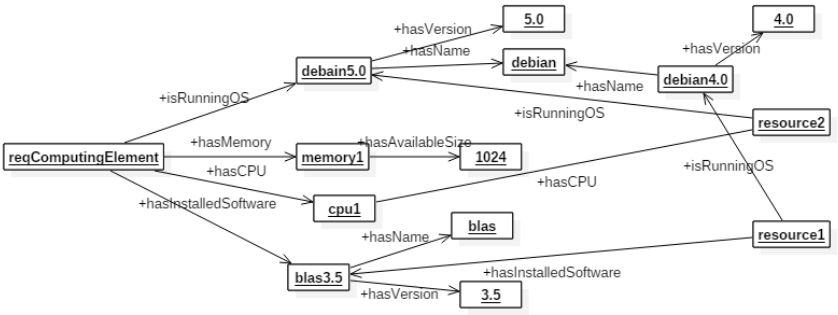

FIGURE 4: Fragment of individuals model

Multicriterial analysis

Finally, we have considered user support based on multicriteria analysis. We have investigated several methods from different categories (*e.g.*, MUAT, outranking, *etc.*) having in mind that (i) all knowledge (resources, knowledge domain, expert opinions) in the system is ontologically represented; (ii) multiple criteria with different weights should be considered; (iii) structure of requirements description is different (subsets of criteria) but based on the ontology; (iv) not all requirements (criteria) may have values in offers (alternatives). As a result, we have selected the AHP method, as the only one capable of matching these requirements (see, [10]). Note that, the GRIP method ([16, 17]), which is based on holistic pairwise comparisons between alternatives in order to construct partial and complete preorder, was further analyzed. Even though the method provides promising direction for ontology-based MCA, it is current implementation requires to compare alternatives as a whole, *i.e.*, user has to consider all properties. This may be problematic in AiG case because of the number of criteria and their hierarchical structure. Hence, the GRIP was also abandoned.

The Analytical Hierarchy Process (AHP; [18, 19, 20, 21]), is a multicriterial analysis for complex problems with multiple, antagonistic and subjectively assessed criteria (with hierarchical structure) originating from multiple experts. Verbal scale allows assessment of qualitative and quantitative criteria. Even though there is some controversy regarding this method, *e.g.*, rank reversal, it turned out that it takes greatest advantage of hierarchical problem structure.

Note that, in the SPARQL query and the class expression matching, the Grid user is assumed to know precisely what she wants. Next, in the graph-based matchmaking, user was assumed to have knowledge necessary to properly assign priorities to resource properties concerning the needed resource configuration. Here, AHP-based MCA approach is intended for users that may have limited knowledge about resource configuration. Its is assumed that they specify "as much as they know" and the system should help them defining resource requirements for their problem and verify that the proposed solution method is consistent with suggestions made by domain experts.

Assuming that the user has limited knowledge, we can distinguish three cases: (i) user does not know much how to solve the problem; (ii) user has wrong ideas how to solve the problem; (iii) different experts have different opinions on the best way to solve the problem. Here, the system should provide the user with recommendations on how to modify / extend / make more precise the requirements. Henceforth, expert opinions need to be "aggregated" so that a "representative opinion" is selected and confronted with user provided "problem specification."

Note that the strength of the MCA approach comes from use of captured expert domain knowledge (captured in an ontology; for details see [22]). Domain ontology defines concepts related to three aspects of domain: problems to be solved, algorithms/methods to solve them, objects that these algorithms/methods operate on. Additional concepts needed in user support are: domain expert, expert opinion (domain expert + recommended resource), job profile (expert opinion + problem / algorithm / objects). Here, opinions have strengths corresponding to weights assigned to experts based on their expertise in the domain.

The job execution scenario with the MCA is as follows:

1. User specifies requirements for needed resource(s) for the job (if he has such knowledge) and the "job profile" – problem and optionally input data characteristics and preferred machine/method/algorithm to be used to solve the problem (this information is to be matched with the expert recommendations);
2. *User_agent* searches in expert knowledge repository, selects pertinent expert opinions and, using AHP, chooses the one that is most representative (AHP is used to evaluate each expert opinion "from the point of view" of each expert; automated selection of opinion that is "closest" to all opinions for a given problem);
3. Three situations are possible
   - user fully and "correctly" defined requirements – and no modifications are needed;
   - resource and job specification could use some "improvement" (*e.g.*, more detailed resource description could be provided – and such are to be suggested by the systems);
   - resource and/or job specification are "wrong" (alternatives are to be proposed by the system).
4. User accepts or rejects feedback based on the expert knowledge (note that the final decision belongs to the user);
5. The remaining part of the scenario follows the general one (in Section a));
6. Note that the contract selection can be also based on the MCA.

Let us consider the following simple example. User wants to commission a task to Grid resource – finding smallest eigenvalue. User knows that the input data for the algorithm is a matrix that is sparse and symmetric. Moreover, he assumes that the best method to solve the problem is to use BLAS library in version 3.5 (requirement for the installed

FIGURE 5: User job specification and restrictions on needed resource

software). User thinks that there should be at least 512 MB of storage space available for the problem data. No other information concerning solution of the problem is provided. This information is provided to the system (see,Figure 5).

For the user input from Figure 5, the following ontology snippet is generated and passed to *user_agent* (note that namespaces where shortened and some parts were omitted to save space):

```
<owl:equivalentClass>
  <owl:Class>
    <owl:intersectionOf rdf:parseType="Collection">
      <rdf:Description rdf:about="cgo#ComputingComponent"/>
      <owl:Restriction>
        <owl:onProperty rdf:resource="aig_exp#hasJobProfile"/>
        <owl:someValuesFrom>
        <owl:Class>
        <owl:intersectionOf rdf:parseType="Collection">
        <rdf:Description rdf:about="aig_exp#JobProfile"/>
        <owl:Restriction>
          <owl:onProperty rdf:resource="aig_exp#forData"/>
          <owl:someValuesFrom>
            <owl:Class>
            <owl:intersectionOf rdf:parseType="Collection">
            <rdf:Description rdf:about="aig_exp#Matrix"/>
            <owl:Restriction>
            <owl:onProperty rdf:resource="aig_exp#hasProperty"/>
            <owl:hasValue rdf:resource="aig_exp_inst#sparseMatrixProperty"/>
            </owl:Restriction>
            <owl:Restriction>
            <owl:onProperty rdf:resource="aig_exp#hasProperty"/>
            <owl:hasValue rdf:resource="aig_exp_inst#symmetricMatrixProperty"/>
            </owl:Restriction>
            </owl:intersectionOf>
            </owl:Class>
          </owl:someValuesFrom>
        </owl:Restriction>
        <owl:Restriction>
          <owl:onProperty rdf:resource=aig_exp#forProblem"/>
__________<owl:hasValue_rdf:resource="aig_exp_inst#smallestEigenvalueProblem"/>
________</owl:Restriction>
________</owl:intersectionOf>
________</owl:Class>
________/owl:someValuesFrom>
______</owl:Restriction>
______<owl:Restriction>
________<owl:onProperty_rdf:resource="aig#hasInstalledSoftware"/>
________<owl:someValuesFrom>
__________...
________</owl:someValuesFrom>
______</owl:Restriction>
______<owl:Restriction>
________<owl:onProperty_rdf:resource="aig#hasStorageSpace"/>
________<owl:someValuesFrom>
__________...
________</owl:someValuesFrom>
______</owl:Restriction>
____</owl:intersectionOf>
__</owl:Class>
```

```
</owl:equivalentClass>
```

Let us assume that the following expert opinions, that match the job profile specified by the user, are available in experts ontology. The code snippet shows individuals defined in the system ontology that represent job profiles, for which experts have entered recommendations in the past and that match user restrictions (note that namespaces were shortened to save space):

```
<owl:NamedIndividual rdf:about="exp_inst#eigenvalueProblemJobProfile1">
  <rdf:type rdf:resource="exp#JobProfile"/>
  <exp:hasExpertOpinion rdf:resource="exp_inst#Opinion1ForEigenvalueProblem"/>
  <exp:hasExpertOpinion rdf:resource="exp_inst#Opinion2ForEigenvalueProblem"/>
  <exp:forProblem rdf:resource="exp_inst#smallestEigenvalueProblem"/>
  <exp:withAlgorithm rdf:resource="#exp_inst#qrFactorization"/>
  <exp:forMatrix rdf:resource="exp_inst#sparseSymmetricMatrix"/>
</owl:NamedIndividual>

<owl:NamedIndividual rdf:about="exp_inst#eigenvalueProblemJobProfile2">
  <rdf:type rdf:resource="exp#JobProfile"/>
  <exp:hasExpertOpinion rdf:resource="exp_inst#Opinion3ForEigenvalueProblem"/>
  <exp:forProblem rdf:resource="exp_inst#smallestEigenvalueProblem"/>
  <exp:withAlgorithm rdf:resource="#exp_inst#JacobiMethod"/>
  <exp:forMatrix rdf:resource="exp_inst#sparseSymmetricMatrix"/>
</owl:NamedIndividual>
```

Job profile individuals have properties that indicate that they consider finding smallest eigenvalue problem for a sparse and symmetric matrix. Two expert recommendations exists that indicate QR factorization and Jacobi method as an algorithm to be used to solve the problem. The following code snippet shows definition of expert opinions that were referenced in the previous code snippet. Expert *HPCExpert1* has recommended resource *resource1*, expert *HPCExpert2* has recommended resource *resource2* and *HPCExpert3* resource *resource3*.

```
<owl:NamedIndividual rdf:about="exp_inst#Opinion1ForEigenvalueProblem">
  <rdf:type rdf:resource="exp#ExpertOpinion"/>
  <exp:hasExpert rdf:resource="exp_inst#HPCExpert1"/>
  <exp:hasRecommendedResource rdf:resource="aig_inst#resource1"/>
</owl:NamedIndividual>

<owl:NamedIndividual rdf:about="exp_inst#Opinion2ForEigenvalueProblem">
  <rdf:type rdf:resource="exp#ExpertOpinion"/>
  <exp:hasExpert rdf:resource="exp_inst#HPCExpert2"/>
  <exp:hasRecommendedResource rdf:resource="aig_inst#resource2"/>
</owl:NamedIndividual>

<owl:NamedIndividual rdf:about="exp_inst#Opinion3ForEigenvalueProblem">
  <rdf:type rdf:resource="exp#ExpertOpinion"/>
  <exp:hasExpert rdf:resource="exp_inst#HPCExpert3"/>
  <exp:hasRecommendedResource rdf:resource="aig_inst#resource3"/>
</owl:NamedIndividual>
```

Note that *resource1*, *resource2* and *resource3* are individuals representing resources already registered in the system ontology or instances of class expressions that specify restrictions on a resource. The latter allows experts not only to select resource from the system as the one that is recommended for a given problem but also to specify what the wanted resource specification should look like. The *User_agent* uses information from the system ontologies to support user in resource selection and optionally in algorithm/method selection. The process of applying the AHP to select the recommendations is described in [20, 21]. Briefly, the system evaluates each expert opinion matching job profile restrictions assessing it "from the point of view" of each expert that entered this opinion in the system, and selects expert opinion that is closest to all opinions for a given job profile. In the example described above, let us assume that *HPCExpert1* and *HPCExpert2* have higher priorities than *HPCExpert3* and the *Opinion2ForEigenvalueProblem* is chosen. This opinion suggests using the QR factorization from the BLAS library (*Opinion1ForEigenvalueProblem* could have suggested EIGEN library) and the recommended resource should have size of available virtual memory greater than 1024 MB. Information about recommended changes / extensions to the requirements is returned to the user. If the user accepts them, the following restriction is added to user requirements specified via web interface:

```
<owl:Restriction>
  <owl:onProperty rdf:resource="aig#hasMemory"/>
    <owl:someValuesFrom>
      <owl:Class>
        <owl:intersectionOf rdf:parseType="Collection">
        <rdf:Description rdf:about="aig#VirtualMemory"/>
        <owl:Restriction>
          <owl:onProperty rdf:resource="aig#hasAvailableSize"/>
          <owl:someValuesFrom>
          <rdfs:Datatype>
          <owl:onDatatype rdf:resource="XMLSchema#integer"/>
          <owl:withRestrictions rdf:parseType="Collection">
          <rdf:Description>
          <xsd:minExclusive rdf:datatype="XMLSchema#integer">1024<xsd:minExclusive>
          </rdf:Description>
          </owl:withRestrictions>
          </rdfs:Datatype>
          </owl:someValuesFrom>
       </owl:Restriction>
      </owl:intersectionOf>
    </owl:Class>
  </owl:someValuesFrom>
</owl:Restriction>
```

If there are no matching expert opinions for a given job profile no recommendations are returned to the user.

Modified user requirements are sent the CIC component, and as in earlier approaches, resources matching the requirements are returned to the *user_agent*. If there are no resources matching the requirements then user is given an appropriate information and needs to modify the requirements.

Resources returned by the CIC are evaluated with the AHP procedure as described in [21]. In the system, there are available default and user-defined comparison matrices that allow to calculate local and global priorities for ontology properties (corresponding to evaluation criteria). Each returned resource specification is evaluated as it was described in [20] and a numerical value (score) is assigned to it. User is presented with list of resources ranked according to the score from AHP. The higher the value the better the resources matches user requirements.

In this way, the expert knowledge, captured and stored in the system, can be used to help the user to correctly and precisely formulate her requirements for job execution.

CONCLUSIONS

In this paper we have considered ways of supporting user in multicriterial selection of resources in computational Grids. The main contribution of this work is to systematically match the techniques from semantic data processing with real-world scenarios materializing in Grid use.

Let us now generalize results of our investigations. Let us assume that in a system that provisions resources all information is ontologically represented. Here, let us first assume that the user is "all-knowing." In this case the needed support consists of finding out if / which resources match exactly her needs. This allows use of simple methods based on SPARQL or sematic reasoners. After user specifies what he/she needs, he/she receives a list of resources that exactly match his/her request. This list is unordered and there is no way to express importance of individual criteria. If none of available resources matches the specified criteria exactly, the request will "fail."

When the graph-based method for semantic matchmaking is used, it is possible to capture expert knowledge concerning importance of relations between concepts and importance of individual criteria to the user. The resulting list of available resources is ordered according to semantic closeness to the user request. Thus, this approach is not likely to "fail" (in the sense of giving no suggestion at all), as all individuals, in a common ontology, are linked to each-other. Here, it is assumed that user has knowledge allowing him to express individual preferences.

Finally, combining the AHP method with semantic technologies allows the system to "make suggestions." Considering multiple expert opinions, allows the decision support subsystem to make the user request more detailed, or to suggest that a completely different method should be used. Here, the main problem is (as in many similar situations) how to capture and keep up-to-date the expert knowledge.

An interesting question that arises, and will be investigated, is: how to "recognize" that the user needs support and the MCA should be applied (instead of exact semantic matchmaking).

ACKNOWLEDGEMENTS

Work presented here is a part of the Poland-Bulgaria collaborative grant "Parallel and distributed computing practices." It is also a part of Poland-Romania collaboration "Software agents and agent systems." Finally, it is also a part of the Serbia-Romania-Poland collaboration "Agent systems and applications."

REFERENCES

[1] M. Dominiak,W. Kuranowski, M. Gawinecki, M. Ganzha, and M. Paprzycki, "Utilizing agent teams in grid resource managementpreliminary 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, "Forming and managing agent teams acting as resource brokers in the Gridpreliminary considerations," 2008, pp. 9–16.

[3] K. Wasielewska, M. Drozdowicz, M. Ganzha, M. Paprzycki, N. Attaui, D. Petcu, C. Bădică, 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 P*roceedings of the First International Symposium on Computing and Networking – Across Practical Development and Theoretical Research*, 2013.

[5] A. Vazhenin, Y. Watanobe, K. Hayashi, M. Drozdowicz, M. Ganzha, M. Paprzycki, K. Wasielewska, and P. Gepner, "Agent-based resource management in tsunami modeling," *in Proceedings of the 2013 Federated Conference on Computer Science and Information Systems*, IEEE Press, Los Alamitos, CA, 2013, pp. 1047–1052.

[6] M. Drozdowicz, K. Wasielewska, M. Ganzha, M. Paprzycki, N. Attaoui, I. Lirkov, R. Olejnik, D. Petcu, and C. Bădică, "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. Iványi and B. Topping, Saxe-Coburg Publications, Stirlingshire, UK, 2011, pp. 335–354.

[7] Ontologia w ujeciu T. Grubera, `http://www.inzynieriawiedzy.pl/ontologie/definicje`, 2013, [Online; accessed 07-November-2013].

[8] P. Szmeja, K.Wasielewska, M. Ganzha, M. Drozdowicz, M. Paprzycki, S. Fidanova, and I. Lirkov, "Reengineering and extending the Agents in Grid Ontology," in *Large-Scale Scientific Computing*, *Lecture Notes in Computer Science* **8353**, edited by I. Lirkov, S. Margenov, and J. Waśniewski, Springer, 2014, pp. 565–573.

[9] M. Drozdowicz, M. Ganzha, M. Paprzycki, P. Szmeja, and K. Wasielewska, "Ontoplay – a flexible userinterface for ontology-based systems," in *AT*, 2012, pp. 86–100.

[10] K. Wasielewska, M. Ganzha, M. Paprzycki, C. Badica, M. Ivanovic, and I. Lirkov, "Multicriteria analysis of ontologically represented information," in *AIP CP***1629**, edited by M.D. Todorov, American Institute of Physics, Melville, New York, 2014, pp. 281–295.

[11] Sparql query language for rdf, `http://www.w3.org/TR/rdf-sparql-query/`, 2015, [Online; accessed 27-August-2015].

[12] Resource description framework (rdf), `http://www.w3.org/RDF/`, 2015, [Online; accessed 27-August-2015].

[13] Web ontology language (owl), `http://www.w3.org/2001/sw/wiki/OWL`, 2015, [Online; accessed 27-August-2015].

[14] S. Rhee, J. Lee, M. Park, M. Szymczak, G. Frackowiak, M. Ganzha, and M. Paprzycki (2009) *Fundam. Inform.* **96**, 395–418.

[15] M. Szymczak, G. Frackowiak, M. Ganzha, M. Paprzycki, S. Rhee, J. Lee, Y. Sohn, J. Kim, Y. Han, and M. Park, "Ontological matchmaking in a duty trip support application in a virtual organization," in *Proceedings of the International Multiconference on Computer Science and Information Technology, IMCSIT 2008, Wisla, Poland, 20-22 October 2008*, 2008, pp. 243–250.

[16] J. Figueira, S. Greco, and R. Slowinski (2009) *European Journal of Operational Research* 460–486.

[17] S. Greco, R. Slowinski, J. Figueira, and V. Mousseau, in *Trends in Multiple Criteria Decision Analysis*, *International Series in Operations Research & Management Science*, Vol. **142**, edited by M. Ehrgott, J. R. Figueira, and S. Greco, Springer US, 2010, pp. 241–283.

[18] T. Saaty, *The Analytic Hierarchy Process*, RWS Publications, Pittsburg, 1990.

[19] T. Saaty (2008) Decision making with the Analytic Hierarchy Process, *International Journal of Services Sciences*, 83–98.

[20] K.Wasielewska and M. Ganzha, "Using analytic hierarchy process approach in ontological multicriterial decision making – Preliminary considerations," in *AIP CP***1487**, edited by M.D. Todorov, American Institute of Physics, Melville, New York, 2012, pp. 95–103.

[21] K.Wasielewska, M. Ganzha, M. Paprzycki, P. Szmeja, M. Drozdowicz, I. Lirkov, and C. Bădică (2014) "Applying Saaty's Multicriterial Decision Making Approach in Grid Resource Management," *Information Technology and Control*, 73–87.

[22] K. Wasielewska, M. Ganzha, M. Paprzycki, and I. Lirkov, "Developing ontological model of computational linear algebra – preliminary considerations," in *AMiTaNS'13, AIP CP***1561**, edited by M.D. Todorov, American Institute of Physics, Melville, New York, 2013, pp. 133–144.