INSTYTUT BADAŃ SYSTEMOWYCH
POLSKIEJ AKADEMII NAUK

TECHNIKI INFORMACYJNE TEORIA I ZASTOSOWANIA

Wybrane problemy
Tom 3 (15)

poprzednio

ANALIZA SYSTEMOWA W FINANSACH
I ZARZĄDZANIU

Pod redakcją
Andrzeja MYŚLIŃSKIEGO

Warszawa 2013

INSTYTUT BADAŃ SYSTEMOWYCH
POLSKIEJ AKADEMII NAUK

TECHNIKI INFORMACYJNE
TEORIA I ZASTOSOWANIA

Wybrane problemy
Tom 3 (15)

poprzednio

**ANALIZA SYSTEMOWA W FINANSACH
I ZARZĄDZANIU**

Pod redakcją
Andrzeja Myślińskiego

Warszawa 2013

Wykaz opiniodawców artykułów zamieszczonych
w niniejszym tomie:

Dr hab. inż. Maria GANZHA, prof. PAN

Prof. dr hab. inż. Zbigniew NAHORSKI

Dr hab. Marcin PAPRZYCKI, prof. PAN

Prof. dr hab. inż. Andrzej STRASZAK

Prof. dr hab. inż. Stanisław WALUKIEWICZ

ISBN 9788389475442

REENGINEERING AND IMPLEMENTATION OF ADAPTABLE AGENTS IN GRID ONTOLOGY

Katarzyna Wasielewska & Paweł Szmeja

Studia Doktoranckie IBS PAN,
e-mail: *katarzyna.wasielewska@gmail.com, pawel.szmeja@gmail.com*

Abstract.. The context for the paper is given by *Agents in Grid* (*AiG*) project, which aims at the development of an agent-based infrastructure with semantic data processing for efficient resource management on the Grid. All information required by the system is ontologically demarcated, therefore, a set of ontologies was developed and/or adopted to represent respective data. Advances in ontology engineering that have been made over the years since inception of the *AiG* warranted a revision of the applied ontologies. This was done in order to modernize and bring them up to the emerging ontology engineering standards as well as apply experience gained throughout development of *AiG*. The aim of this paper is twofold. First, to discuss the process and issues encountered while reenginering existing *AiG* ontologies and approach undertaken to design a hierarchy of ontologies that is extensible and adaptable in case of many deployments. Second, to show how lessons learned were applied to create from scratch the ontology of domain knowledge. Domain ontology was initially created for the area of computational linear algebra, however, its structure was designed to be flexible enough to model arbitrary domain of knowledge.
Keywords: ontology engineering, Agents in Grid, AiG, domain knowledge

1 INTRODUCTION

The context for this paper is provided by *Agents in Grid* project (*AiG*; [2, 5, 8]) aiming at development of a flexible agent-based infrastructure, which is to facilitate intelligent resource management in the Grid. The proposed approach is based on application of semantic data processing in all aspects of the system, therefore, all knowledge is stored in ontologies, while communication protocols utilize messages with ontological content ([4]). The first ontologies that were developed provided concepts necessary to describe: (i) resources and Grid structure, (ii) contract requirements for *Service Level Agreement* negotiations, and (iii) content of messages that are exchanged in the system. Moreover, the ontology that models Grid structure was developed on the basis of the previously existing

Core Grid Ontology (*CGO*; [9]; 108 concepts, 57 properties). The original *CGO* was extended (72 additional concepts, 27 additional properties) and modified to match the needs of the *AiG* project (see, [4, 8]) and serve as a top level ("base") ontology. During the extension, features identified as most problematic were modified as a method of troubleshooting problems that appeared as the development of *AiG* progressed. At this stage analysis of how every modification would impact all the other ontologies was not performed. Furthermore, conditions (29 additional concepts, 45 additional properties) and messaging (14 additional concepts, 3 additional properties) ontologies have been created and incorporated into existing hierarchy of ontologies. The next step in the ontology development was the addition (development from scratch) of a new domain knowledge ontology (currently 288 concepts, 29 properties) that is part of decision support provided to the user. Fig. 1 shows hierarchy of ontologies utilized in the system.

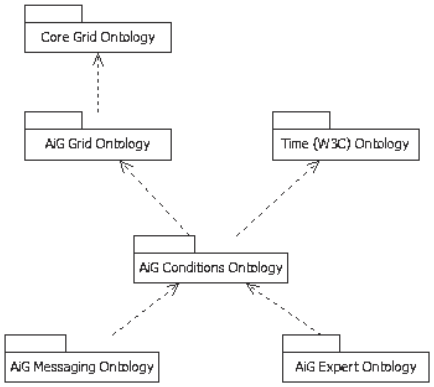

Fig. 1. Ontologies in AiG system

With increasing complexity of the ontological structures and gradual move from simple to advanced reasoning (i.e. including hundreds of RDF triples), we were faced with recurring errors generated by reasoners. Those included mostly inferences that were not intended and unexpected (but still a result of a sound reasoning) rather than those resulting in straight up unsatisfiable or inconsistent ontologies. With further system development and testing of the set of *AiG* ontologies, new needs arose to: (i) develop a new domain knowledge ontology that would not duplicate problems al-

ready existing in utilized ontologies, (ii) redesign structure of the ontology hierarchy so that multiple deployments with slightly different knowledge (in different Grid systems) would be supported. Ontology reengineering became a necessity to alleviate and ultimately reduce the ontology related problems to a minimum.

2 AiG ONTOLOGY REENGINEERING

In the process of reengineering of *AiG* ontologies the entire ontology hierarchy was examined. Most of the work revolved around the top level ontologies (*CGO* and *AiG Grid Ontology*[1]). The base ontologies have generally the most impact on *AiG*. Lower level ontologies import and use concepts defined in upper levels, so any change on the top is essentially propagated down the hierarchy. Since the hierarchy was built from the top (i.e. top level ontologies are the oldest), the newer (lower) ontologies (created with more experience) required little maintenance, which further merited the focus on the ontological base.

2.1 Documentation standards

In any group effort documentation is an essential part of development. This is especially important in case of ontologies, because, according to the standards and best practices, they are meant to be reusable and interconnected. Proper meta-information makes ontologies more intuitive and assures that the meaning of ontological concepts is clear. In OWL ([10]), this can be achieved through proper documentation, clear naming scheme, and overall consistency. After examination we have found out that the original *CGO* had problems in those areas and some of them carried over into *AiG Grid Ontology*.

General ontology engineering standards state that names of OWL $classes$ should be capitalized, whereas OWL $property$ names should start with a lowercase letter, preferably in the format of *"has[Property]"* or *"is[Property]"*. This is particularly important for a hierarchy of ontologies, because naming schemes carry over to all ontologies that import a given ontology. If ontologies in a hierarchy use different naming conventions, the overall naming scheme is broken. Original CGO contained many entities that did not conform to any standardized naming scheme. Being a top level ontology any inconsistencies in naming carried over into all other

[1] Please note that "*AiG Grid Ontology*" refers to a specific ontology file within the hierarchy of *AiG* ontologies and is in no way synonymous with a set of all ontology files in *AiG*

AiG ontologies. An example is the *operatingSystem* property that not only conflicted the naming scheme of "is/has" prefixes (the scheme was applied to some other properties such as *hasCPU* and *hasFileSystem*), but also could be easily confused with the $OperatingSystem$ class. Recall that in OWL, IRIs should be unique (i.e. one IRI can describe no more than one entity). In the scope of one ontology it means that IRI fragment needs to be unique. Since IRI does not contain any information about type of OWL entity (i.e. $property$, $class$ or $individual$) the class of $OperatingSystem$ and property of $operatingSystem$ have IRIs that differ only by the capitalized (or not) letter "O". Such naming made the ontology error-prone as it was very easy for a programmer to make a small mistake in the IRI. Considering that a reasoner treats the asserted knowledge as undisputed facts, mistaking $OperatingSystem$ with $operatingSystem$ would have critical and unforeseen consequences for the operation of entire system. To solve the problem, the property has been renamed to *hasOperatingSystem*. The remaining (similar) problems have also been fixed.

OWL annotations are a very useful documentation tool. Original *CGO* was sparsely annotated which made it difficult to know what usage the authors intended for each of the entities. Concepts that did not have a suggestive name, comments or usage ended up not being used, because there was no indication of the reason for their existence or what the authors intended for them. We have updated every ontology in the hierarchy with annotations such as general comments and usage tips, versioning information and entity labels (with language tags) in the hopes of decreasing required maintenance and improve general understanding of the ontology. *AiG* ontologies are constantly being updated with annotations that are to serve both as documentation for developers and guidelines for the users that are interested in technical details of applied ontologies. As part of further development we plan to use annotations in the dynamic user interface (see, [3] for more details). Here, the GUI, in addition to dynamically adjusting to the ontology structure, would also display information taken directly from OWL annotations to provide users with clear entity labels and tooltips (possibly multilingual) explaining how to best use the system.

2.2 Ontology hierarchy

In the *AiG* the ontological base is comprised of *CGO* and *AiG Grid Ontology* that imports it, both regarded as upper ontologies. In the scope of the *AiG* those are viewed as being on the same (topmost) level in the hierarchy. For this reason when examining how the reengineering process would

impact the hierarchy *AiG Grid Ontology* and *CGO* were considered (conceptually) to be a single ontology. We examined how the upper concepts would behave when extended (imported) into lower levels and came to the conclusion that some of them were not suitable for a top-level ontology.

An example of how the original *CGO* was unsuitable for extension was the *clockSpeed* property. It's original use, in the *CGO*, is summarized by two constructs: the restriction on the *CPU* class, and the domain specification on itself. The first defines that every CPU needs defined property *clockSpeed*. The latter restricts the *clockSpeed* to the CPUs only. In the *AiG Grid Ontology* we introduced the *GPU* class that, just like CPU, had to be described by the clock speed. Because of the domain restriction it was impossible to use the *clockSpeed* property from the *CGO*. Any GPU that used this property would be inferred to be a CPU. While formally sound, such inference was against our intentions. Originally to avoid changing the *CGO* file, a *hasClockSpeed* property was introduced in the *AiG Grid Ontology*. There, it had the same interpretation as the *clockSpeed* from the *CGO*, only with the GPU, as well as the CPU, in its domain.

This is an example of a concept that is "too specific" for an upper ontology. Putting a restriction on the property that is intended to represent a clock speed we come to a false conclusion that only CPUs can have a clock speed. It would not be a mistake in an ontology that deals specifically with CPUs. In the Grid, however, we come across GPUs, and possibly APUs or specialized (e.g. physics simulation) processing units, so in the context of a Grid ontology clock speed should be applicable to various classes. Specifically, its use should not be limited in the top level of ontology hierarchy.

While reengineering, the domain restriction on *CGO* property *clockSpeed* was removed. Next *AiG Grid Ontology clockSpeed* property was moved to *CGO* and finally renamed to *hasClockSpeed*. The class definitions for both CPUs (defined in *CGO*) and GPUs (defined in *AiG Grid Ontology*) contain a requirement to have a *hasClockSpeed* property with an integer value. By removing the unnecessary restrictions from top level ontologies we made it easier to use the upper concepts in lower level ontologies. Other such problems with different properties were resolved in a similar fashion.

2.3 Cleaning conceptual inconsistencies

Careful examination of *AiG* ontologies revealed that a number of errors has accumulated in them as a result of bugs in used and/or developed software (that have been since fixed). One such group of errors were "du-

plicates" of entities. Those were entities with the same IRI fragment and representing the same concept, but defined in different ontologies. An example of such error was the *CPU* class that was defined both in *CGO* and in *AiG Grid Ontology*. The definitions differed and, although when combined they provided full description of CPUs, separately neither of them was complete (e.g. only one had the *hasClockSpeed* property requirement). Consequently results of reasoning about either of those classes was never what we expected, despite being technically complete. That, in turn, led to problems with inferring class hierarchy or classifying the ontologies. Note that, these errors became apparent only after reasoners started to be used in a working system on the full-blown ontology (400+ entities) rather than on mini-examples (10-20 entities) used in testing the agent infrastructure. Definitions of both *CPU* classes were merged and moved into *CGO*. The declaration of *CPU* was removed from *AiG Grid Ontology*. Other problems of this type were dealt with in the same way. As a result of this operation the reasoning problems (unexpected results) were eliminated.

We have also found inconsistencies in definition of entities and their usage. For example the original *CPU* class had a restriction of *"hasClockSpeed only string"* in the definition which contradicted with the *hasClockSpeed* property range which was an integer. Such formulation was formally correct, but it rendered the *hasClockSpeed* property useless for CPUs. The *CPU* definition was changed to accept integer values on *hasClockSpeed* property. Other inconsistencies included seemingly random usage of some properties in definitions. For instance, a *hasName* property had been put in definitions of many classes, but with varying restrictions. Those ranged from any of existential, type, quantifier, or universal restriction or any combination of them. Since there was no indication as to why some classes required a "name" specifically of type *string*, others could only have one "name", while some lacked any restrictions on *hasName*, we decided to substitute the varying *hasName* restrictions with a set of consistent ones. Now, any entity that by definition needs a name, has exactly one *hasName* of type string. After examining definitions in every *AiG* ontology we removed any inconsistencies. Even though those were not errors and sometimes didn't impact the work of the system at all, removing them made the ontology seem cleaner and more concise.

2.4 Reengineering summary

Let us note that, ongoing research and literature concerns ontology merging, alignment, mapping, but almost never concerns "software engineering

like" principles for ontology re-use (see, for instance [1]). This being the case let us summarize most important lessons learned from our work.

When working with multiple ontology files one should be mindful of the existing (or planned) ontology hierarchy, and how a new/modified ontology would fit into it. Hierarchies can vary but it's always good to remember that upper ontologies should contain as most as possible "general concepts" and avoid introducing unnecessary conditions that would restrict usage of upper entities. Consequently, the hierarchy level should be reflected in the level of ontological specialization, when moving deeper into the imports chain. Special attention should be given to upper ontologies, because the higher the level, the more ontologies are affected via imports.

We should always remember that ontologies are meant to be shared and reused. Thus, it is crucial to clearly communicate their intended use (e.g. by providing complete annotations and adhering to the naming standards). This can help in prevention of misuse of a concept or (re)defining it more times than intended. As the developed ontology grows, standards and consistency in naming, annotating and definitions become an important tool in prevention of human mistakes or unexpected and unwanted inference results. They also reduce maintenance time and increase understanding and intuitiveness of an ontology.

Finally, applying an ontology *in practice* is indispensable for identifying the problems that exist in its design and helps to understand the importance of developed standards and best practices.

3 FURTHER AiG ONTOLOGY RESTRUCTURING

The complexity of ontologies used in the system as well as logical division of their scope lead to modularization. Ontologies form a hierarchy where each conceptual level can contain multiple ontology files. Note that in an OWL ontology we can distinguish three different types of statements[2]: (i) T-Box (terminology) – describes conceptualization i.e. axioms defining and describing the classes, (ii) A-Box (assertions) – contains assertions about the instances in the domain, (iii) R-Box (roles) – technically a subset of A-Box, it consists of facts about roles such as inverse, property chain etc. A-Box and T-Box statements together form a knowledge base. The initial structure of *AiG* ontologies is presented in Fig. 1. During the development process, it became necessary to redesign and restructure ontologies in a way that would support multiple deployments. Through experiments we have found that there is a clear distinction between the roles

[2] this division comes directly from description logic (on which OWL is based)

of A-Box and T-Box in the context of a Grid system. Very often one conceptual model (T-Box) is common for all deployments i.e. concepts related to resource descriptions and possible properties are the same in all Grid environments where the system would run. On the other hand, instances and their assertions (A-Box) are specific for a given deployment i.e. description of resources available in a given Grid is realized with A-Box statements. Therefore, decision was made to split the *AiG* ontology hierarchy into two branches – one containing the conceptual model and the other – instances. From each ontology file we extracted A-Box facts and put them in a separate file, while leaving T-Box in the original one. Resulting files were put in a hierarchy that mirrors the one existing before and joined with the old files using "horizontal" imports (i.e. imports realized on the same conceptual level). The resulting ontology hierarchy presenting the import structure is shown in Fig. 2.

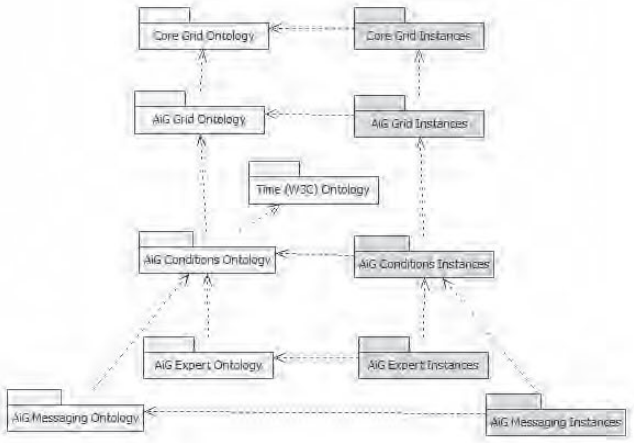

Fig. 2. Ontologies in AiG system after restructuring

The set of ontologies collectively named *Instances* (with the "Instances" postfix in the name) is treated as deployment specific. *AiG* system components first load T-Box ontologies (*Core Grid Ontology*, *AiG Grid Ontology*, *AiG Conditions Ontology*, *AiG Messaging Ontology*, *AiG Expert Ontology*), and then, depending on system startup parameters, required instance ontologies. It should be noted, that namespaces used in instance ontologies are different for each ontology (different for e.g. *Core Grid Instances*,

AiG Grid Instances, *AiG Conditions Ontology*), however, in order to avoid unnecessary complication of the process of ontology loading, for each deployment specific ontology for a corresponding conceptualization (T-Box) ontology the namespace is the same e.g. for *AiG Grid Instances* ontology it should be the same for deployment in any environment. The constraint of such approach is that for each hierarchy level only one ontology file with instances can be loaded. We have made an assumption that in most cases every deployment would need only one set of ontologies and different deployments do not have any connection with each other, so there is no possibility of IRI conflict. This assumption can be relaxed by setting the system startup parameter identifying deployment to a set of proper namespaces of multiple instance ontologies on a given level.

4 DESIGN OF NEW ADAPTABLE DOMAIN ONTOLOGY

During the development of *AiG* system the need arose to add a new (created from scratch) ontology that would provide means to achieve goals described in [6, 7]. Specifically, *AiG* system is to provide support beyond the functionalities found in existing Grid middlewares i.e. help the user to choose optimal algorithm and/or resource to solve a given problem utilizing ontological representation of domain knowledge. Computational linear algebra was chosen as an initial domain to be modeled. The most crucial concepts that the domain ontology should specify are: problems, algorithms/methods to solve them, and objects on which algorithms/methods operate. The ontology under development is included in ontology hierarchy presented in Fig. 2 and is extending the existing *AiG* ontologies. Moreover, while creating the *AiG Expert Ontology* experiences and conclusions from reengineering were taken into account.

The main goal of the new 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. Initially, the expert ontology was developed for a single domain of knowledge – computation linear algebra. However, we believe that the selected approach will enable the *AiG* system to also work with other domains e.g. differential equations. Therefore, apart from identifying concepts common for description of any area of knowledge, the naming conventions that were applied should be easily adaptable for arbitrary domain e.g. class initially named *MatrixProperty* was renamed to *DataProperty*, property *forMatrix* was renamed to *forData*. Proper naming does not limit the understanding of data to only matrices, since in different domain, objects on which algorithms operate

may be of different types. The main concepts included in *AiG Expert Ontology* that are used to model knowledge from one or more domains are (see Fig. 5):

- *Domain* – identifies the area of knowledge that is modeled. Other concepts in the ontology can be related to a given domain with the *hasDomain* property.
- *Problem* – hierarchy of problems from a given domain (see Fig. 3). Instances of class *Problem* can point to a domain in which they are defined with *hasDomain* property.

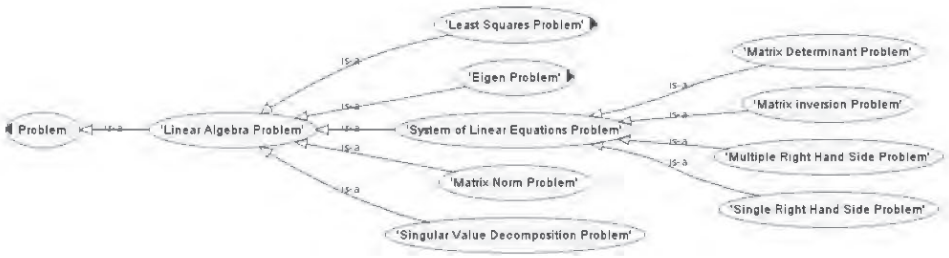

Fig. 3. Part of hierarchy of problems for computational linear algebra in *AiG Expert Ontology*

- *Algorithm* – algorithms/methods that can be used to solve problems from a given domain. Instances of class *Algorithm* can point to a domain in which they can be applied with *hasDomain* property.

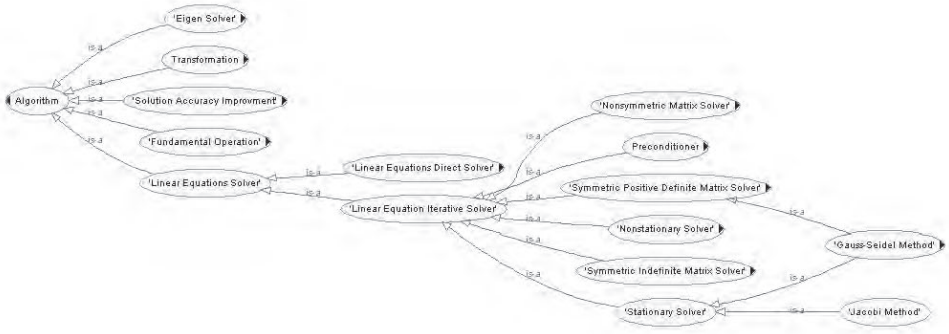

Fig. 4. Part of hierarchy of algorithms for computational linear algebra in *AiG Expert Ontology*

– *Data Element* – type of input data (objects) that algorithms/methods operate on e.g. simple, structured.
– *Data Property* – the hierarchy of properties that can characterize input data. For computational linear algebra properties are grouped into subclasses of *MatrixProperty* and *MatrixElementProperty*. Instances of *Data Element* and *Data Property* classes are related with *hasProperty* object property.
– *Data Storage Format* – file format in which input data is stored. Class is a range for *hasStorageFormat* property, which domain is *DataElement*. For computational linear algebra example formats are: CSV, MAT, MTX.
– *Domain Expert* – concept representing experts (people or computer system) that provide recommendations within a given domain.
– *Job Profile* – concept that with respective properties relates instances of *Problem*, *Data Element* and *Algorithm* classes with instances of *Expert Opinion* class. Class expression is specified by the user to describe what problem does she want to solve, and what is her knowledge about input data and algorithm (not all properties have to be used e.g. user may know problem and general properties of input data but has no knowledge about possible algorithms). The class expression is then matched with job profiles that are available in *AiG Expert Ontology*.
– *Expert Opinion* – concept that relates instances of *Domain Expert* and *Grid Entity* classes i.e. expert with the resource specification that she recommends for solving a specific problem. Obviously, resources originate from *AiG Grid Ontology*.

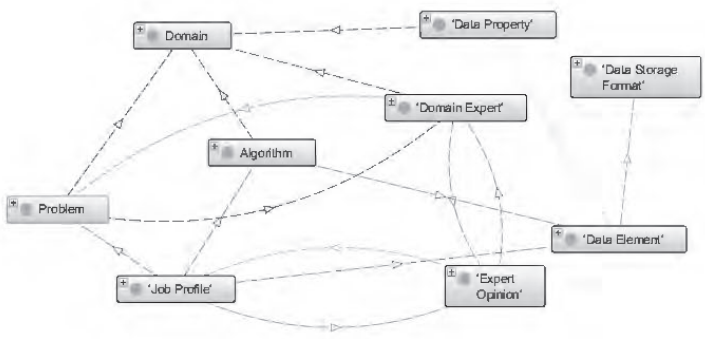

Fig. 5. Relations between concepts from *AiG Expert Ontology*

It can be easily 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. class *Problem* at the moment has one subclass *Linear Algebra Problem*, however, one can add another subclass e.g. *Differential Equations Problem* and its possible further decomposition into more specific problems. Similarly, new subclasses can be defined for *Algorithm* and *Data Property* classes (see Fig. 6 for sample ontology scheme with generic class naming). Obviously, for the division of concepts between domains, new instances of *Domain* class should be defined and assigned as value to *hasDomain* property of respective instances. Alternatively, ontologies for different domains can be stored in separate OWL ontology files, but the top level concepts and properties will be common.

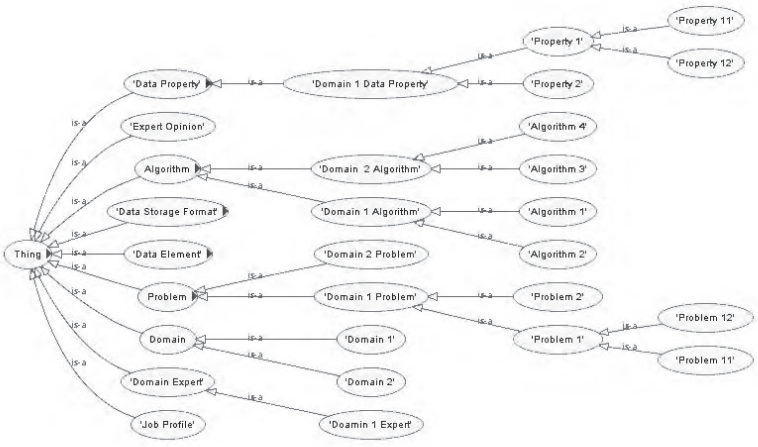

Fig. 6. Sample generic ontology structure in *AiG Expert Ontology*

In the *AiG Expert Ontology* earlier specified guidelines for ontology engineering were followed, e.g. naming conventions for classes and properties, filling annotations for ontology elements. The following example shows definition of *Chebyshev Iteration* class. Each new class has a defined label that is to be displayed in the user interface taking into account also language attribute. Moreover, classes have comments that indicate their meaning and possible usage. This information can be as well displayed as tooltip in the user interface.

```
<owl:Class rdf:about="&AiGExpertOntology;ChebyshevIteration">
    <rdfs:label xml:lang="en">Chebyshev Iteration</rdfs:label>
```

```
    <rdfs:subClassOf rdf:resource="&AiGExpertOntology;
        NonstationarySolver"/>
    <rdfs:subClassOf rdf:resource="&AiGExpertOntology;
        NonsymmetricMatrixSolver"/>
    <rdfs:comment xml:lang="en">The Chebyshev Iteration
        recursively determines polynomials with coefficients
        chosen to minimize the norm of the residual in a min–max
        sense.</rdfs:comment>
</owl:Class>
```

5 CONCLUDING REMARKS

The aim of this paper was to present lessons learned from the process of ontology reengineering and restructuring based on *AiG* ontologies. Secondly, the experiences gained during the modernization and extension of *AiG* ontologies, were applied to develop a new adaptable and easily extensible ontology to be used within the *AiG* system. Our next goal is to further develop adaptable domain knowledge ontology of computational linear algebra as well as other selected domain and apply it in the user decision support within the *AiG* system with the functionality of switching between domains.

References

1. Fensel Dieter. *Ontologies: A Silver Bullet for Knowledge Management and Electronic Commerce*. Springer-Verlag, New York, 2003.
2. Mateusz Dominiak, Wojciech Kuranowski, Maciej Gawinecki, Maria Ganzha, and Marcin Paprzycki. Utilizing agent teams in Grid resource management—preliminary considerations. In *Proc. of the IEEE John Vincent Atanasoff Conference*, pages 46–51, Los Alamitos, CA, 2006. IEEE CS Press.
3. Michał Drozdowicz, Maria Ganzha, Katarzyna Wasielewska, Marcin Paprzycki, and Paweł Szmeja. Using ontologies to manage resources in grid computing: Practical aspects. In Sascha Ossowski, editor, *Agreement Technologies*, volume 8 of *Law, Governance and Technology Series*, pages 149–168. Springer Netherlands, 2013.
4. Michał Drozdowicz, Katarzyna Wasielewska, Maria Ganzha, Marcin Paprzycki, Naoual Attaui, Ivan Lirkov, Richard Olejnik, Dana Petcu, and Costin Badica. *Ontology for Contract Negotiations in Agent-based Grid Resource Management System*. Saxe-Coburg Publications, Stirlingshire, UK, 2011.
5. Wojciech Kuranowski, Maria Ganzha, Maciej Gawinecki, Marcin Paprzycki, Ivan Lirkov, and Svetozar Margenov. Forming and managing agent teams acting as resource brokers in the grid—preliminary considerations. *International Journal of Computational Intelligence Research*, 4(1):9–16, 2008.
6. Michael Lucks. *A Knowledge-Based Framework for the Selection of Mathematical Software*. PhD thesis, Southern Methodist University, 1990.

7. Dana Petcu and Viorel Negru. Interactive system for stiff computations and distributed computing. In *Proceedings of IMACS'98: International Conference on Scientific Computing and Mathematical Modelling*, pages 126–129. IMACS, 1998.
8. Katarzyna Wasielewska, Michał Drozdowicz, Maria Ganzha, Marcin Paprzycki, Naoual Attaui, Dana Petcu, Costin Badica, Richard Olejnik, and Ivan Lirkov. Trends in Parallel, Distributed, Grid and Cloud Computing for engineering. chapter Negotiations in an Agent-based Grid Resource Brokering Systems. Saxe-Coburg Publications, Stirlingshire, UK, 2011.
9. Wei Xing, Marios D. Dikaiakos, Rizos Sakellariou, Salvatore Orlando, and Domenico Laforenza. Design and Development of a Core Grid Ontology. In *Proc. of the CoreGRID Workshop: Integrated research in Grid Computing*, pages 21–31, 2005.
10. OWL 2 Web Ontology Language. `http://www.w3.org/TR/owl2-overview/`.

Dostosowanie oraz implementacja elastycznych ontologii w ramach systemu Agents in Grid

Streszczenie.

Kontekst dla artykułu stanowi projekt *Agents in Grid* (*AiG*), którego celem jest rozwój infrastruktury do wydajnego zarządzania zasobami na Gridzie w oparciu o agenty programowe oraz semantyczne przetwarzanie danych. Wszystkie informacje wykorzystywane przez system AiG reprezentowane są ontologicznie, dlatego też, na potrzeby przechowywania istotnych danych, został zaimplementowany lub zaadaptowany zestaw ontologii. Ze względu na postępy w obszarze inżynierii ontologii, które nastąpi}ly w latach od rozpoczęcia prac nad *AiG*, rewizja istniejących ontologii stała się koniecznością. Prace miały na celu modernizację oraz uaktualnienie ontologii zgodnie z pojawiającymi się standardami dotyczącymi inżynierii ontologii, jak również korekty wynikającej z doświadczeń, zdobytych podczas implementacji systemu. Cel tego artykulu jest dwojaki. Po pierwsze, omówienie procesu oraz problemów napotkanych podczas dostosowywania istniejących w systemie *AiG* ontologii oraz nowego podejścia do zmiany struktury ontologii, która ułatwi dalszą rozbudowę oraz przystosowanie w przypadku wielu wdrożeń. Po drugie, pokazanie w jaki sposób wnioski wyciągnięte podczas zmian w istniejących ontologiach zastosowano przy tworzeniu od podstaw nowej ontologii, przechowującej wiedzę domenową. Ontologia domenowa została początkowo opracowana dla obszaru obliczeniowej algebry liniowej, natomiast jej struktura została zaprojektowana w sposób umożliwiający modelowanie wiedzy z różnych dziedzin.

ISBN 83-894-7550-2